DENIO DUARTE

# Utilizando Técnicas de Programação Lógica Indutiva para Mineração de Banco de Dados Relacional

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre. Curso de Pós-Graduação em Informática. Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientadora: Prof ª. Dr ª. Silvia Regina Vergilio

Co-Orientadora: Profª. Drª. Aurora T. R. Pozo

CURITIBA – PR

2001

## Resumo

As empresas estão sendo sobrecarregadas rapidamente com grandes volumes de dados e ao mesmo tempo estão se tornando, predominantemente, orientadas ao conhecimento. O aumento do domínio do conhecimento não melhora apenas os produtos, mas também é uma fonte para decisões estratégicas. Do ponto de vista da ciência da computação, os requisitos de conhecimento exigidos pelas empresas sempre dão mais ênfase a "conhecer que" (conhecimento declarativo) do que "conhecer como" (conhecimento procedural). A lógica matemática tem sido a representação preferida para o conhecimento declarativo e portanto, técnicas de descoberta do conhecimento são utilizadas, as quais geram fórmulas lógicas a partir dos dados. Programas lógicos oferecem uma representação poderosa e flexível para restrições, gramáticas, equações e relacionamentos temporais e espaciais. A técnica que induz conceitos a partir de dados gerando programas lógicos é chamada de Programação Lógica Indutiva (ILP – Inductive Logic Programming).

Este trabalho descreve a implementação de um sistema para a descoberta do conhecimento (mineração de dados) em bancos de dados relacionais utilizando fundamentos de ILP e SQL. Esse sistema, DBILP (***D**ata**B**ase miner based on **ILP***), trabalha com dois mecanismos básicos: um que, baseado em ILP, especializa e, em seguida, generaliza as regras construídas, e outro que instância e valida essas regras, baseado em comandos SQL. ILP e SQL foram escolhidos pois o primeiro é uma técnica relativamente moderna e expressiva para a mineração de dados, e o segundo permite que grande volume de dados sejam manipulados, graças ao controle feito por um Sistema Gerenciador de Banco de Dados (SGBD).

O funcionamento do DBILP é discutido ao longo desse trabalho, apresentando os módulos que o compõe, a sintaxe da linguagem de entrada definida, e a forma que as regras são construídas dentro do espaço de busca.

A eficiência do DBILP é apresentada através de um experimento utilizando três outros sistemas bem conceituados no meio acadêmico, dois orientados a atributo-valor: C4.5 e CN2; e um sistema ILP: Progol. A análise desse experimento indica que o DBILP é particularmente útil no processo de descoberta do conhecimento em banco de dados (KDD – Knowledge Discovery in Databases).

## Abstract

Industry is increasingly overwhelmed by large-volume-data. Industry is also becoming predominantly knowledge driven. Increased understanding not only improves products, but strategic decision making. From Computer Science point of view, the knowledge requirements within industry often give higher emphasis to “knowing that” (declarative knowledge) rather than “knowing how” (procedural knowledge). Mathematical logic has always been the preferred representation for declarative knowledge and thus knowledge discovery techniques are required which generate logical formulae from data. Logic programs provide a powerful and flexible representation for constraints, grammars, equations and temporal and spatial relationships. The technique that induces concepts from data building logic programs is called Inductive Logic Programming (ILP).

This work describes a system implementation for knowledge discovery (data mining) in relational databases using ILP e SQL techniques. This system, DBILP (**D**ata**B**ase miner based on **ILP**) works with two basic engines: the first one, based on ILP, specializes and, afterwards, generalizes the built rules; the second one grounds and validates those rules, using SQL statements. ILP and SQL were chosen because the former is a new and expressive technique to mine data and the latter allows DBILP to work with a large data volume, since Database Management Systems (DBMS) controls data access and manipulation.

Throughout this work DBILP mecanisms are described, its modules presented, input language sintax defined, and the way the system builds the rules in search space is described.

DBILP’s effectiveness is shown by an experiment using three well-known systems, two attribute-value oriented: C4.5 and CN2; and a ILP system: Progol. The experiment analysis points out that DBILP is particularly well suited for Knowledge Discovery in Databases (KDD) tasks.

# Índice

## Definições Utilizadas

Completeness = completude

Background Knowledge = conhecimento prévio do domínio

To subsume = subjugar

□ = vazio

Prior Satisfiability = satisfação anterior

Posterior Satisfiability = satisfação posterior

Prior Necessity = necessidade anterior

Posterior Sufficiency = suficiência posterior

Minimality = minimalidade

Inverse Resolution = resolução inversa

Using Rule Models = utilização de modelo de regras

Lattice = reticulado

Bias = tendência

~ = negação

To prune = podar

## Lista de Figuras

## Lista de Tabelas

# Capítulo 1

## 1 Introdução

### 1.1 Contexto

Atualmente as empresas estão sob grande pressão para responder, rapidamente, às mudanças do mercado. Para realizar esta tarefa, as empresas devem ter acesso rápido a todos os tipos de informações antes de tomar qualquer decisão estratégica. Para possibilitar a tomada de decisão correta é essencial poder pesquisar em registros passados e identificar tendências relevantes. Obviamente, para executar qualquer tarefa de análise de tendência deve-se ter acesso a todas informações que possam servir de apoio. Estas informações estão armazenadas, principalmente, em banco de dados volumosos. Neste cenário, a maneira mais fácil de obter acesso aos dados que geram as informações para a tomada de decisão é criar um *data warehouse* [ADR97]

*Data warehouse* é um repositório de dados central que contém todos os dados relevantes de uma organização para a tomada de decisão. O mesmo é projetado para apoiar decisões estratégicas, através dos sistemas de apoio à decisão (SAD), e é construído a partir das bases de dados dos sistemas operacionais da organização. Normalmente, os dados dos sistemas operacionais podem estar em formatos, locais e plataformas diferentes. Estas características das fontes dos dados operacionais faz com que a criação e a manutenção dos *data warehouses* sejam muito complexas. Isto se dá pois os dados das fontes têm que ser capturados, limpos, integrados e aplicados no *data warehouse*. Além de passarem pelos pré-processamentos descritos, os dados deste repositório têm as seguintes características [DEV97]:

1. Históricos, i.e., as cada atualização gera um novo registro com a data e hora que esta atualização ocorreu;
2. Não-voláteis, i.e., um registro não é alterado/excluído;
3. Orientados ao assunto, i.e., as informações armazenadas no *data warehouse* se referem a um assunto de interesse para a empresa; e
4. Integrados, i.e., consolidam todos os dados oriundos de fontes diferentes.

Como os *data warehouses* armazenam informações que podem ser estratégicas para as empresas, tornam-se um ambiente ideal para ser aplicado um processo chamado descoberta do conhecimento em banco de dados (KDD – *Knowledge Discovery in Databases)*.

O objetivo do KDD é a descoberta de informações úteis a partir de grandes coleções de dados. Essa descoberta pode ser regras que descrevam as propriedades dos dados, freqüência de ocorrência de um determinado padrão, agrupamento de objetos em um banco de dados, entre outros. O processo de KDD se divide em vários passos: compreender o domínio e preparar os dados (passos que já estão feitos em um *data warehouse*), descobrir padrões, pós-processar os padrões descobertos e colocar os resultados em uso. O passo que descobre os padrões é conhecido como *data mining*[1]. Esse passo é de particular interesse nesse trabalho.

A atividade de KDD é bastante desafiadora, principalmente na aplicação de *data mining*, pelas seguintes razões[MOR97]:

- A quantidade de tuplas (registros) em um banco de dados para um algoritmo aprender pode exceder ao número de exemplos (casos) que algoritmos de aprendizado ou procedimentos estatísticos conseguem manipular;
- A quantidade de atributos em banco de dados pode exceder ao número de características diferentes que são normalmente trabalhadas por algoritmos de aprendizado; e
- A tarefa de encontrar todas as regras válidas e escondidas em um banco de dados é mais difícil que em outros ambientes.

Existem vários algoritmos de aprendizados para realizar *data mining*: Classificadores Bayesianos[WEI91] e Redes Neurais[ADR97], Indução através de Árvores de Decisão[UBE00], Aprendizado de Regras de Associação[ADR97], Algoritmos Genéticos[WEI98], entre outros. Para serem aplicados ao processo de KDD, esses algoritmos adaptam o banco de dados em uma das seguintes maneiras:

- A análise é feita em um conjunto de atributos equivalentes com valores binários; ou
- A análise é feita em uma tabela que compreende a maioria dos atributos interessantes.

Além de restringirem os banco de dados, a maioria dos algoritmos restringem também a descoberta do conhecimento, pois a representação é feita através de atributo-valor (lógica proposicional). Isto pode ser uma restrição severa pois exclui o conhecimento estrutural ou relacional. S. Dzeroski[DZE96] apresenta um exemplo que ilustra bem essa situação. A Figura 1.1

[1] Alguns autores/pesquisadores utilizam o termo KDD e *data mining* intercambiavelmente. Neste trabalho serão tratados distintamente, segundo Mannila[MAN97].

apresenta um banco de dados relacional que possui duas tabelas: *Potential-Customer* que armazena dados sobre potenciais clientes de uma empresa hipotética, e a tabela *Married-To* que representa casais. Um algoritmo de aprendizado, que é restrito ao conhecimento proposicional, poderia aprender as seguintes regras[2]:

*income(Person) > =100000 --> customer(Person)=yes*

*sex(Person)=f & age(Person)>=32 --> customer(Person)=yes*

As seguintes regras não poderiam ser expressas (consequentemente aprendidas) por esses algoritmos (por não poderem expressar relacionamentos):

*married(Person,Spouse) & customer_yes(Person) --> customer_yes(Spouse)*

*married(Person,Spouse) & income(Person,>=100000) --> customer_yes(Spouse)*

| Person | Age | Sex | Income | Customer |
|---|---|---|---|---|
| Ann Smith | 32 | F | 10 000 | yes |
| Joan Gray | 53 | F | 1 000 000 | yes |
| Mary Blythe | 27 | F | 20 000 | no |
| Jane Borwn | 55 | F | 20 000 | yes |
| Bob Smith | 30 | M | 100 000 | yes |
| Jack Brown | 50 | M | 200 000 | yes |

Potential-Customer

| Husband | Wife |
|---|---|
| Bob Smith | Ann Smith |
| Jack Brown | Jane Borwn |

Married-To

Figura 1.1 – Tabelas Exemplos

Outras relações não podem ser expressas ou caracterizadas pela lógica proposicional, incluindo relações espaciais, relações temporais, ou conectividade em um grafo. Para este tipo de aprendizado, a lógica de primeira ordem é necessária[MOR97].

A Programação Lógica Indutiva (ILP – Inductive Logic Programming) é um campo dentro do aprendizado de máquina que investiga o aprendizado de hipóteses em lógica de primeira ordem. Assim ILP é considerada uma das técnicas mais eficazes para a descoberta de padrões pois a esma oferece uma forma poderosa para representar restrições, gramáticas, planos, equações e

[2] Essas regras foram transformadas para o formato lógico mas mantido o poder de expressão proposicional. Na fórmula proposicional, a primeira regra seria: *se income >= 10000 então customer=yes.*

relacionamentos temporais. As linguagens de programação lógica, tal como Prolog, usam um sub-conjunto da lógica matemática para oferecer um linguagem rica para representação declarativa. Enquanto a maioria das técnicas de aprendizado de máquina fazem a aquisição do conhecimento de modo procedural, ILP oferece uma forma declarativa de aquisição utilizando exemplos, conhecimento prévio do domínio e hipóteses [MUG97].

O aprendizado de um programa ILP é definido como: dado um conhecimento prévio do domínio *B*, expresso com um conjunto de predicados, exemplos positivos E+ e negativos E-, encontrar uma fórmula de predicado lógico *H*, tal que [BRA98]:

1. todos os exemplos em E+ podem ser logicamente derivados de B^H, e
2. nenhum exemplo negativo em E- pode ser logicamente derivado de B^H.

Esta definição é similar ao problema genérico do aprendizado indutivo, mas insiste na apresentação do conhecimento no formato de predicados de primeira ordem B e H.

Tradicionalmente, a mineração de dados é realizada utilizando abordagens orientadas à representação de atributo-valor (lógica proposicional), podendo ser citados os sistemas CN2 e C4.5[CLA91, QUI93].

A maioria dos sistemas ILP atuais, tais como Progol, FOIL, Golem, FILP [MUG00,QUI90,MUG90,BER96], entre outros, apresenta a mesma deficiência: carregam os dados a serem processados para a memória principal do computador (RAM – Random Access Memory). Esta característica faz com que esses sistemas tenham limitações para trabalhar com grande volume de dados, já que a memória RAM é limitada, além de ser compartilhada com outros aplicativos. Existem poucos trabalhos que aplicam ILP em conjunto com banco de dados relacionais [BLO96, BRO96,SHI97].

## 1.2 Motivação

Dado o contexto acima, são apresentadas, a seguir, as principais motivações para a realização desse trabalho de pesquisa:

- ILP é uma técnica relativamente nova e pode ser utilizada para mineração em banco de dados com as seguintes vantagens:
    1. Pode utilizar conhecimento prévio do domínio para auxiliar no processo de busca de padrões;

2. Como usa a lógica de predicados fica fácil trabalhar com múltiplas tabelas pois permite representar relacionamentos entre objetos;

3. Possui um poder de expressão maior que técnicas orientadas a atributo-valor.

- SQL pode ser empregada conjuntamente com ILP para mineração de dados, facilitando a manipulação de grandes volumes de dados;
- A possibilidade de ILP e SQL serem utilizados para induzir conceitos; e
- Existem poucos trabalhos que utilizam fundamentos de ILP para minerar banco de dados relacional e a maioria tem problemas para gerenciar os dados pela limitação da memória RAM.

### 1.3 Objetivos

Este trabalho tem como objetivo investigar a utilização de técnicas ILP para minerar banco de dados relacional. Como resultado desse estudo um sistema, DBILP (***D**ata**B**ase miner based on **ILP***) é proposto. O sistema objetiva a mineração de bancos com grande volume de dados.

Um dos aspectos importantes na mineração de dados é o forma de trabalhar com o espaço de busca. DBILP possui uma linguagem de entrada que permite implementar duas características importantes para a redução do espaço de busca em ILP: a tendência declarativa e a de preferência. A primeira permite definir o espaço de hipóteses a ser considerado pelo aprendiz (o que procurar), e a segunda, determina como procurar e qual hipótese focar no espaço definido.

O DBILP trabalha com os dados armazenados em tabelas controladas por um sistema gerenciador de banco de dados (SGBD). Esta característica evita que o DBILP se preocupe com a forma que os dados estão armazenados ou com a maneira de acessá-los pois a maioria dos SGBD oferecem uma linguagem de manipulação de dados (DML - *Data Manipulation Language*) para estas finalidades. No caso de banco de dados relacionais, esta DML é implementada através da linguagem SQL[DAT89]. Assim o sistema proposto pode ser aplicado em ambiente com grande volume de dados, como em *data warehouses*, no processo de KDD.

### 1.4 Organização

Este capítulo apresentou o contexto no qual esse trabalho está inserido, a motivação para realizá-lo e o objetivo a ser atingido.

No Capítulo 2 são introduzidos a terminologia e os principais conceitos relacionados à mineração de dados.

O Capítulo 3 descreve com mais detalhes a Programação Lógica Indutiva pois esses conceitos são utilizados no trabalho realizado.

No Capítulo 4 são apresentados os sistemas estudados que se baseiam nas técnicas de ILP e que serviram como base para o DBILP.

O Capítulo 5 descreve os detalhes de implementação do DBILP. É apresentado o funcionamento, como foi implementado, além de um exemplo mostrando passo a passo uma execução.

No Capítulo 6 são apresentados os resultados relacionados ao experimento utilizado para comparar as regras geradas pelo DBILP com três outros sistemas: um que utiliza ILP, e outros dois que utilizam técnicas de classificação orientadas a atributo-valor.

O Capítulo 7 apresenta as conclusões, contribuições e perspectivas de trabalhos futuros.

O trabalho contém um apêndice que apresenta os dois algoritmos mais importantes do DBILP.

# Capítulo 2

## 2 Terminologia e Conceitos Básicos

Neste capítulo são introduzidos os conceitos utilizados para o desenvolvimento desse trabalho. As técnicas de *data warehousing* e *data mining* são descritas, e seus principais aspectos de implementação e utilização são discutidos.

### *2.1 Data Warehousing*

Atualmente, informação é essencial no ambiente organizacional. O sucesso está intimamente ligado com o uso da mesma de forma antecipada e decisiva. A utilização das informações de uma organização se faz a partir dos dados que ela possui e o volume destes cresce dia a dia, dificultando a busca de informações. Poucas organizações possuem uma fração das informações que elas realmente precisariam, apesar do grande volume de dados que existe. A distinção entre dado e informação é fundamental para enfrentar os problemas que uma organização encontra[DEV97].

Para oferecer informações sobre os dados, os analistas de sistemas criaram os Sistemas de Apoio à Decisão (SAD). Tais sistemas, inicialmente, não eram satisfatórios para os níveis estratégicos de uma organização para auxiliar na tomada da decisão[DEV97,INM97]. Toda a análise de dados e tomada de decisão era baseada em sumários ou relatórios derivados dos sistemas operacionais da empresa. Dificilmente havia cruzamento dos dados. Essas ferramentas respondiam apenas à questões do tipo [OLI98]:

. Quantos clientes temos?

. Quanto vendemos no ano passado?

. Qual é o estoque do produto *x*?

Os SAD ou Sistemas de Informações tradicionais, geralmente limitados no escopo e flexibilidade, oferecem acesso aos dados pelos usuários finais. São baseados em programas de cópias feitos pelo departamento de sistemas de informação que copiam e customizam os dados dos sistemas operacionais, de acordo com as necessidades desses usuários. Algumas abordagens acessam dados diretamente das bases de dados operacionais que, freqüentemente, estão espalhadas pela empresa em vários formatos, ou fazem cópias de pequenas partes para o seu processamento. A

utilidade dessas abordagens é limitada pois as consultas são demoradas por competirem com os acessos dos sistemas operacionais[DEV97, INM97].

Esta abordagem para os sistemas informacionais, quando necessitam integrar os dados, executam dois passos genéricos [WID95]:

1. Aceitar a consulta, determinar os conjuntos de fontes de informações apropriadas para responder a consulta, gerar sub-consultas apropriadas ou comandos para cada fonte de informação; e

2. Obter resultados das fontes de informações, executar a tradução apropriada, filtrar, unir as informações e retornar a resposta final para o usuário da aplicação.

Este processo é referido como abordagem *lazy* ou *on-demand* para a integração dos dados, ou seja, a informação é extraída das fontes de dados apenas quando as consultas são ativadas. Esta abordagem não é satisfatória[WID95] pois como apresentado anteriormente, as consultas competem com os sistema operacionais.

Várias técnicas foram criadas para oferecer sistemas informacionais mais completos e confiáveis. Recentemente, um conjunto de novos conceitos e ferramentas evoluíram para um nova tecnologia. Esse conjunto torna possível atacar o problema de prover acesso para tomadores de decisões das empresas a todo o tipo de informação necessária para a tomada de decisões rápidas e com grande grau de acerto [OLI98, DEV97]. O termo que caracteriza esta nova tecnologia é chamado de *data warehousing*, que provê uma alternativa, natural, para a integração dos dados [WID95]:

1. As informações de cada fonte de interesse são extraídas antecipadamente, traduzidas, filtradas apropriadamente e combinadas com informações relevantes de outras fontes, e armazenadas (logicamente) num repositório central.

2. Quando uma consulta é ativada, a mesma é avaliada diretamente neste repositório, sem acessar às fontes originais.

Este repositório integrado é chamado de *data warehouse*. Portanto, *data warehousing* é um conjunto de ferramentas para extrair dados dos sistemas operacionais, limpá-los, integrá-los, convertê-los, agregá-los, enriquecê-los e aplicá-los no *data warehouse*, além de oferecer meios para os usuários consultá-los [DEV97].

A Figura 2.1 apresenta, simplificadamente, as duas abordagens de sistemas de informação, a 2.1(a) representando os sistema tradicionais e a 2.1(b) os sistemas aplicando *data warehousing*. De um lado verifica-se uma abordagem mais simples porém menos eficiente, de outro, uma mais complexa, com necessidade de mais processamento para gerar o repositório central, porém, oferecendo ao usuário os dados mais propícios para consultas e tomadas de decisão [DEV97,INM97].

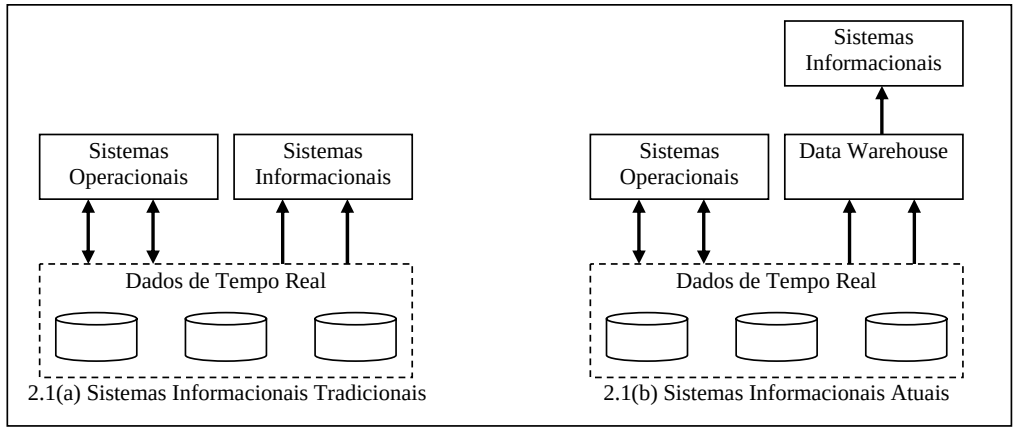



Figura 2.1 - Estrutura dos Sistemas de Informação

### 2.1.1 Abordagens

Existem, na literatura disponível, várias abordagens para estruturar um *data warehousing*. Uma das encontradas é descrita por [DEV97] (Figura 2.2). A mesma é chamada de abordagem em 3 camadas:

- A primeira camada é a camada dos dados de tempo real - os dados do dia-a-dia da organização;
- A segunda camada possui os dados reconciliados, onde os dados dos diversos, heterogêneos e distribuídos sistemas são capturados, combinados e melhorados dentro de um único modelo de dados. O objetivo da abordagem é que seja uma única fonte de dados para todos os usuários finais. Desta camada são derivadas todas as combinações que os usuários necessitarão. Neste processo os dados são limpos para eliminar inconsistências e irregularidades. Nenhum dado é criado neste processo, apenas são preparados para a próxima camada. O usuário final, raramente, talvez nunca, acessa a camada dos dados reconciliados, isto se dá pois, os dados nesta camada estão modelados e estruturados de forma não interessante para o usuário final;

- A terceira camada, chamada camada de dados derivados, é o conjunto de dados que foi montado para as consultas e necessidades mais comuns dos usuários, porém se os usuários necessitarem de consultas que esta camada não puder "responder" a mesma terá que acessar à camada de dados reconciliados.

Independente da abordagem escolhida, o *data warehousing* é empregado em vários setores da indústria: manufaturas (para postar pedidos e suporte a clientes), revenda (para perfis de usuários e inventário), serviços financeiros (para análise de pedidos de empréstimos, análise de riscos, análise de cartão de crédito e detecção de fraudes), transportes (gerenciamento de fluxos), telecomunicações (para análise de chamadas e fraudes), planos de saúde (para análise de requisições), entre outros [CHA96].

Em todas as abordagens estudadas [DEV97, OLI98,CHA96,WID95] é de consenso que o repositório dos dados informacionais deve ficar separado dos dados operacionais. Segundo [CHA96] existem várias razões para tal, entre elas: 1)aplicar processamento analítico on-line (OLAP - *On-Line Analytical Processing*) no *data warehouse*; 2) a funcionalidade e exigências de desempenho dos mesmos são bastante diferentes daquelas exigidas pelo processamento transacional *on-line* (OLTP - *On-Line Transaction Processing*).

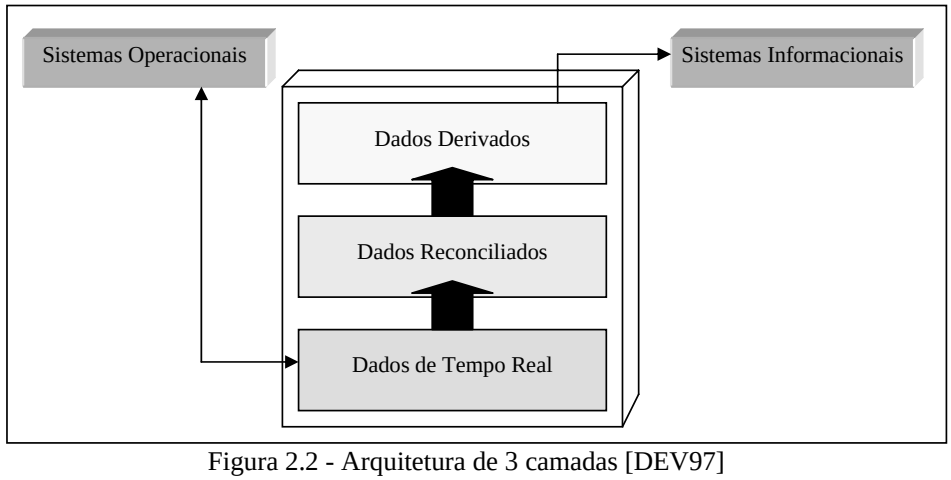



Figura 2.2 - Arquitetura de 3 camadas [DEV97]

### 2.1.2 Processo e Características dos Dados

O processo de copiar os dados operacionais, aqueles que permitem aplicações do tipo OLPT, para o *data warehouse*, envolve uma série de passos [DEV97,CHA96]:

- Extração dos dados das fontes;

- Limpeza dos mesmos, i.e., resolver problemas de inconsistências de tamanhos de campo, de descrição, de valores associados, violação de integridade, entre outros;
- Carga, i.e., colocar os dados no *data warehouse*. São necessários processamentos adicionais: checar restrições de integridade, ordenar, sumarizar, agregar os dados, etc.; e
- Atualização, i.e., após a carga dos dados, os mesmos têm que ser atualizados.

Todos esses passos possuem sub-passos e estes podem ser vistos com mais detalhes em [DEV97].

Os dados no *data warehouse* devem ter algumas características básicas para atender às necessidades dos sistemas informacionais [DEV97]: devem ser persistentes, conter o registro histórico detalhado das transações sobre os dados, consolidados e orientados a assuntos.

A estrutura dos dados no *data warehouse* não é de consenso geral. O que é dito é que a mesma deve, de alguma forma, representar os dados por várias dimensões, onde cada dimensão representa um contexto e a intersecção entre as dimensões são os valores naquele contexto [CHA96]. Existem duas abordagens principais atualmente usadas para construir banco de dados multidimensionais [AGR97]: uma mantém os dados como um cubo de *k*-dimensões baseado numa estrutura especializada de armazenamento não relacional e a outra, usa a abordagem relacional, onde as operações nos cubos são traduzidas para consultas relacionais, e.g., SQL. Esta última utiliza a abordagem estrela ou floco de neve para representar as dimensões[DEV97,CHA96].

A forma que os dados estão organizados em um *data warehouse* é propícia para a aplicação de um passo do processo de descoberta de conhecimento em banco de dados, *data mining* ou mineração de dados.

### *2.2 Data Mining*

*Data mining* ou mineração de dados é a busca de padrões e estruturas em grandes volumes de dados com o objetivo de descobrir informação não explícita. É um esforço compartilhado entre o computador e o homem. O homem projeta bases de dados, descreve o problema e define os objetivos. O computador pesquisa nos dados, procurando por padrões que combinem com os objetivos definidos[MOT97,WEI98]. Segundo [MOT97], o termo minerar está empregado de forma incorreta pois não existe um material em particular para ser minerado, é mais uma “pescaria” de dados, ou seja, é a extração não trivial de conhecimento implícito, antes desconhecido e potencialmente útil, a partir dos dados. Este conhecimento deve ser novo, não óbvio e apto para ser

usado [ADR97]. Isto faz com que um dos maiores desafios na mineração dos dados seja definir corretamente as classes de padrões que possam ser de interesse [MOT97].

Uma comparação feita por [ADR97] para demonstrar a importância da mineração de dados utiliza o que foi escrito por Jorge Louis Borges sobre uma biblioteca infinita – *The Library of Babel*. Esta biblioteca hipotética possui um conjunto infinito de livros, localizados em redes de salas sem fim. A maioria desses livros não possuíam sentidos e seus títulos não eram inteligíveis. Alguém poderia percorrer esta biblioteca a vida toda sem encontrar algo informacional. Esta biblioteca, na realidade, representava um amontoado de dados sem nenhuma informação, e este é o problema que encontramos nos dados hoje: cada vez crescem mais e proporcionalmente, conseguimos extrair menos informação. Então nos defrontamos com um paradoxo, quanto mais dados menos informação.

Na tentativa de resolver este tipo de problema, muitos pesquisadores na área de Inteligência Artificial (IA) e Banco de Dados tentaram encontrar um paradigma para uma forma de extrair informações dos dados. Assim surgiu *data mining*. A partir dos anos 90, as pesquisas na área de mineração de dados tiveram um crescimento visível no setor de pesquisa e de desenvolvimento[MAN97].

Na realidade, *data mining* é parte de um processo maior, chamado descoberta de conhecimento em base de dados (KDD), apesar de alguns autores utilizarem o termo intercambiavelmente. Neste trabalho, *data mining* será considerado como parte do processo KDD. Neste contexto, H. Mannila[MAN97] define que o descobrimento de conhecimento a partir dos dados deveria ser visto como um processo de vários passos:

1. compreender o domínio;
2. preparar o conjunto dos dados;
3. descobrir padrões (*data mining*);
4. pós-processar os padrões descobertos, e
5. colocar os resultados em uso.

O processo de KDD é necessariamente interativo: os resultados do passo de *data mining* podem mostrar que algumas mudanças deverão ser feitas no passo de preparação do conjunto de dados, o pós-processamento dos padrões pode causar uma mudança leve no tipo de padrão, entre outros. O processo de KDD não é uma técnica nova, na realidade é uma área multi-disciplinar, onde

o aprendizado de máquina, estatística, tecnologia de banco de dados, sistemas especialistas e visualização de dados fazem suas contribuições. A Figura 2.3 representa esta associação.

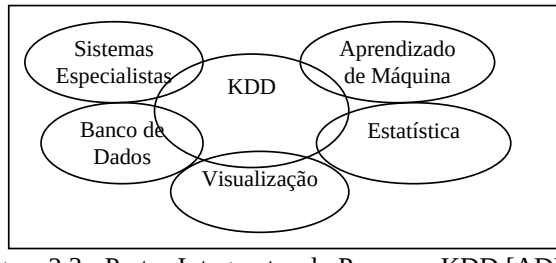



Figura 2.3 - Partes Integrantes do Processo KDD [ADR97]

### 2.2.1 Técnicas e Algoritmos

As técnicas de *data mining* encontram os padrões e relações através da construção de modelos [TWC99]. Modelos, como mapas, são abstrações da representação da realidade. Um mapa pode modelar a rota de um aeroporto até sua casa, mas o mesmo não representa um acidente, que implica em um tráfico lento ou em uma construção que exige um desvio. Enquanto não se pode confundir um modelo com a realidade, um bom modelo é útil como um guia para compreender o negócio e sugerir ações que podem ser tomadas para ajudar a mantê-lo.

Existem, basicamente, dois modelos/métodos de *data mining*, o preditivo e o descritivo (descoberta do conhecimento) [TWC99, WEI98]. O primeiro baseia-se, simplesmente, em que os dados são coletados durante o tempo e as respostas corretas (às vezes descobertas depois) são armazenadas com os casos anteriores. O objetivo é encontrar padrões nos dados que dão respostas corretas aos novos casos. O segundo descreve os padrões nos dados existentes, os quais podem ser usados como guia para as decisões. A diferença fundamental entre estes dois tipos é que o método preditivo faz uma predição explícita, enquanto o descritivo é usado para ajudar a construir o primeiro ou faz uma predição implícita quando elas formam a base para uma ação ou decisão. A ênfase relativa de previsão ou descrição varia com o sistema de *data mining* utilizado. Estes objetivos são atingidos por vários algoritmos. Esses algoritmos implementam várias técnicas de *data mining* [ARU96]

As técnicas de *data mining* podem ser classificadas conforme a função que executam ou de acordo com a classe de aplicação. As principais técnicas são [UBE00]:

1. Classificação: dado um conjunto de registros com seus atributos correspondentes, um conjunto de classes e uma associação de cada classe com os registro, esta técnica examina o conjunto de registros e produz descrições das características dos registros

para cada classe. Quando aprendendo as regras de classificação, o sistema tem que encontrar as regras que predizem a classe dos atributos de predição. Uma vez definidas as classes, o sistema inferirá regras que governam a classificação, portanto, o mesmo estará apto para encontrar a descrição de cada classe. Uma regra é dita correta se sua descrição cobre todos os exemplos que pertençam à classe e nenhum dos exemplos que não pertençam à classe [UBE00, ARU96]. Na prática uma regra não pode ser 100% correta devido à exceções ou ruídos no dados. As regras descobertas devem prever corretamente as classes de exemplos não vistos durante o treinamento do sistema.

2. Associação ou geração de perfis: dados uma coleção de itens e um conjunto de registros, que contém algum número de itens da coleção, um método de associação é uma operação realizada no conjunto de registros que retornam afinidades ou padrões que existem entre as coleções de itens [UBE00].Estes padrões são expressos por percentuais, e.g., 75% dos clientes que compram um refrigerante do tipo cola também compram batata frita. O número de casos que atendem ao padrão encontrado é chamado suporte, além desta medida é utilizada também a medida de confiança, que é dada pelo número de casos atendidos pelo padrão dividido pelos casos que são atendidos pelo antecedente da regra [ADR97]. Um exemplo típico de aplicação desta técnica é o chamado “cesta de supermercado”, onde são comparados pontos de vendas/horários com os produtos comprados e a associação entre eles.

3. Padrões seqüenciais ou temporais: esta técnica analisa uma coleção de registros em um período de tempo para identificar tendências. Este tipo de algoritmo é especialmente útil para companhias de catálogos e de investimento financeiros, pois ajudam a analisar a seqüência de eventos que afetam os preços dos instrumentos financeiros [UBE00, ARU96].

4. Agrupamento ou segmentação: esta técnica segmenta a base de dados em subconjuntos ou grupos. Estes podem ser criados estatisticamente ou usando rede neurais e métodos indutivos simbólicos não supervisionados. Um grupo é um conjunto de objetos agrupados devido a similaridades e proximidades. A chave é traduzir alguma medida intuitiva de similaridade para uma medida quantitativa. A técnica tem que descobrir subconjuntos de objetos com relação ao conjunto de treinamento e assim encontrar descrições para cada um dos subconjuntos. Ao contrário da classificação, não se sabe a qual grupo um registro pertence [UBE00,TWC99].

Dentro das técnicas apresentadas, vários algoritmos se encaixam, os principais são:

- Ferramentas de consulta (SQL/OLAP): não conseguem encontrar dados escondidos. Alcança 80% das informações, os outros 20% - que podem ser de vital importância – exigem técnicas mais avançadas. Estes tipos de algoritmos fazem o que é chamado de predição ingênua[3], ou seja, o mesmo extrai o conhecimento superficial [ADR97].

- Técnicas de visualização: utilizam o algoritmo de afinidade e distância que faz a metáfora do espaço para determinar a distância entre os registros, aqueles que estão mais próximos são os que têm mais afinidades e, conforme aumenta a distância, menos afinidade terão. Os dados para fazer esta análise devem estar na mesma magnitude. Após os cálculos, pode-se estabelecer os agrupamentos, analisando os dados pelos agrupamentos construídos [ADR97].

- *k*-vizinho mais próximo: analisa os dados por proximidade e vê as tendências dos mesmos. O *k* significa o número de vizinhos a serem analisados. É um algoritmo bastante caro pois faz muitas comparações [ADR97].

- Árvore de decisão: a representação da árvore de decisão é um dos métodos lógicos mais utilizados, e pequenas árvores são relativamente fáceis de entender. Para classificar um caso, o nodo raiz é testado com um ponto de decisão verdadeiro-falso. Dependendo do resultado do teste associado com o nodo, o caso é passado para o próximo ramo, e o processo continua. Quando um nodo folha é encontrado, seu valor armazenado é a resposta. Os caminhos para o nodo folha são mutuamente exclusivos [UBE00].

- Regras de associação: usadas para associar padrões nos dados, e.g., 30% das mulheres com carros esportes vermelhos e cachorros pequenos usam Channel 5. Classicamente são associadas a atributos binários. Como podem ser encontradas milhares de regras que atrapalharão a tomada de decisão, é necessário introduzir algumas medidas para distinguir associações interessantes ou não, além da própria supervisão humana. Tais medidas são: suporte e confiança. Suporte é medido através da verificaçao nos dados de quantos exemplos são atendidos (define-se um percentual desejado) por uma determinada regra, porém esta medida não pode ser a única, pois os dados que confirmam a regra criada em uma determinada situação, podem não confirmar em outra. Confiança é a

[3] näive prediction

medida que compara os exemplos que atendem ao antecendente da regra. Portanto as medidas de suporte e confiança são essenciais para o sucesso desta técnica. Esta técnica é útil quando se tem uma vaga idéia do que se quer [ADR97].

- Rede neurais: são coleções de nodos conectados com entradas, saídas e processamento. São estruturas matemáticas capazes de aprender. O método é resultado de investigações acadêmicas do modelo do sistema nervoso humano. Estes algoritmos têm a habilidade de derivar significado a partir de dados complicados ou imprecisos e podem ser usados para extrair padrões e detectar tendências que são bastante complexas para serem percebidas por humanos ou outras técnicas [ADR97].

- Algoritmos genéticos: basicamente, a análise de dados não é o forte deste tipo de algoritmo[WEI98]. O mesmo pode ser visto como uma solução de vários problemas combinatoriais ou de otimização. O nome do algoritmo deriva da similaridade com o processo natural de seleção.

Neste contexto de mineração de dados, dois sistemas que utilizam técnicas tradicionais para minerar dados serão brevemente mencionados a seguir, CN2 e C4.5. Ambos os sistemas são considerados classificadores.

CN2 [CLA91] é projetado para indução eficiente de regras simples de produção mesmo em ambiente com domínios “com ruído”. Usa um método de busca baseado em diferentes funções de avaliação: *entropy*, *modify*, *entropy*, *laplace* e *naive*. As regras induzidas pelo CN2 são no seguinte formato: *if* <*complex*> *then predict* <*class*>, onde <*complex*> é a conjunção de diferentes testes de atributos. As regras podem ser ordenadas ou não–ordenadas na lista de decisão.

C4.5 [QUI93] é baseado no algoritmo de árvore de decisão. Árvore de decisão é um dos métodos mais simples e que tem mais sucesso para algoritmos de aprendizado[RUS95]. O processo de indução é *top-down*. Para adicionar um ramo na árvore, o algoritmo seleciona o próximo melhor atributo utilizando uma função de avaliação chamada *ganho de informação*. Esta função é projetada para minimizar a profundidade final da árvore. Após isso, a árvore final é transformada em um conjunto de regras.

## 2.3 Considerações Finais

Neste capítulo discutiram-se as principais características de um *data warehouse* e forma de implementar essa técnica. O *data warehouse* foi apresentado como uma fonte de dados própria para ser aplicada as técnicas de *data mining*, que também foram discutidas.

Diversas técnicas de *data mining* têm sido propostas e estudadas com o objetivo de encontrar uma maneira eficaz de extrair dados relevantes de bases de dados. Com o advento do *data warehousing*, os estudos de técnicas e ferramentas que viabilizam a mineração de dados cresceram. Com esses estudos cresceram também as pesquisas para encontrar técnicas que atendam mais eficientemente às necessidades de busca de padrões em base de dados.

No contexto de KDD, os sistemas baseados em atributo-valor (lógica proposicional) são limitados na descoberta do conhecimento, pois existe um grande número de variáveis para manipular; a descoberta envolve múltiplas relações; e o poder de expressão desses sistemas é limitado (exclui o conhecimento relacional).

Permitir que especialistas codifiquem e removam ruídos de seus próprios conhecimentos de um domínio de forma legível é de primária importância no processo de aprendizagem. Assim os sistemas ILP são mais indicados para o processo de KDD pois, além de descobrirem conhecimento em ambiente mais complexos que os sistemas atributo-valor, possuem um poder de expressão maior [MUG97]. ILP será discutido amplamente no próximo capítulo pois é de particular interesse nesse trabalho.

# Capítulo 3

## 3 Programação Lógica Indutiva

Este capítulo apresenta a teoria da programação lógica indutiva (ILP – *Inductive Logic Programming*). Primeiramente, esta técnica é conceituada, em seguida, são apresentadas algumas estruturas para a inferência das regras e, finalmente, é discutido o uso de ILP para a descoberta do conhecimento em banco de dados.

### 3.1 Introdução

Programação Lógica Indutiva é uma combinação do aprendizado indutivo de máquina e a programação lógica (Figura 3.1). Em ILP, a representação baseada na lógica é usada para levar à indução de regras a partir de exemplos. A seguir serão descritas as duas áreas que formam este paradigma [BRA98].

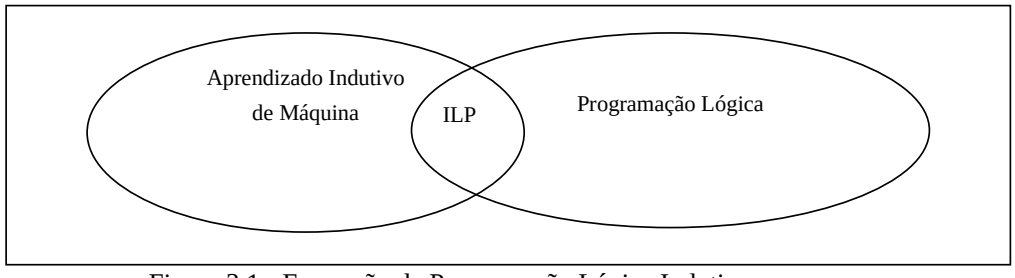



Figura 3.1 - Formação da Programação Lógica Indutiva

#### 3.1.1 Aprendizado de Máquina

O campo de aprendizado de máquina foi concebido há quatro décadas com o objetivo básico de desenvolver métodos computacionais que implementam várias formas de aprendizado, em particular mecanismos capazes de induzir conhecimento a partir de exemplos. Tal forma de indução de conhecimento é particularmente desejável em problemas que não possuem soluções por algoritmos, que sejam mal definidos, ou projetados apenas informalmente. As pesquisas na área de Inteligência Artificial permitiram que problemas não tratáveis pudessem ser resolvidos estendendo a forma tradicional [BRA98] de:

*programa = algoritmo + data*

para:

*programa = algoritmo + dados + conhecimento prévio do domínio (domain knowledge)*

A aplicação do conhecimento prévio do domínio, codificado em estruturas próprias, é fundamental para a resolução de problemas deste tipo. A Figura 3.2 apresenta um esquema para ilustrar o aprendizado de máquina.

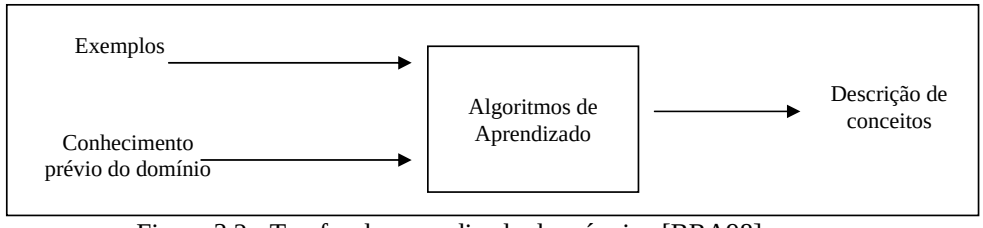



Figura 3.2 - Tarefas do aprendizado de máquina [BRA98]

O aprendizado de máquina indutivo, segundo [RUS95], possui a seguinte estrutura: dado uma coleção de exemplos *e*, retorna uma função *h* que aproxima-se de *e*. A função *h* é chamada hipótese.

### 3.1.2 Programação Lógica

A abordagem declarativa traz muitas vantagens na construção de sistemas inteligentes. A programação lógica tenta estender estas vantagens para todas as tarefas de programação. Qualquer computação pode ser vista como um processo de tornarem explícitas as conseqüências de escolher um programa qualquer para uma máquina em particular e oferecer entradas particulares. A mesma vê o programa e a entrada como um comando lógico em relação ao mundo, e o processo de elaborar as conseqüências explícitas, como um processo de inferência [BRA98].

Uma linguagem de programação lógica possibilita escrever algoritmos através da aplicação das sentenças lógicas com informação para controlar o processo de inferência. *Prolog* é a linguagem de programação lógica mais utilizada atualmente [RUS95].

A Tabela 3.1 apresenta um programa *Prolog* e sua representação na lógica de primeira ordem. Este programa descreve a relação membro, onde é verdadeira se o primeiro argumento é membro do segundo.

Tabela 3.1 – Prolog e Lógica de Primeira Ordem

| **Lógica de Primeira Ordem** | **Prolog** |
|---|---|
| ∀*x,l member(x,[x\|l])* | *member(X,[X\|L]* |
| θ*x,y,l member(x,l) => member(x,[y\|l])* | *member(X,[X\|L]) :- member(X,L)* |

### 3.2 Conceitos Básicos

A Programação Lógica Indutiva está preocupada com o problema de aprender programas lógicos a partir de exemplos [BER96]. O problema de aprender em ILP é normalmente definido como [BRA98]: dado um conhecimento prévio do domínio (*background knowledge*) B, expresso como um conjunto de predicados, exemplos positivos E+ e negativos E-, um sistema ILP construirá uma fórmula de predicado lógico H, tal que:

1. todos os exemplos em E+ podem ser logicamente derivados de B^H, e
2. nenhum exemplo negativo em E- pode ser logicamente derivado de B^H.

Esta definição é similar ao problema genérico do aprendizado indutivo, mas insiste na apresentação de predicados lógicos de B e H.

ILP investiga a construção indutiva de teorias na forma de cláusulas de primeira ordem, conhecidas também como predicados ou cálculo de predicados, a partir de exemplos e conhecimento prévio do domínio. A lógica de primeira ordem oferece uma estrutura formal para descrição e argumentação sobre objetos, as partes, e a relação entre os objetos/partes. Um importante subconjunto da lógica de primeira ordem são as cláusulas de *Horn*. Cláusulas de *Horn* (definidas) são disjunções de cláusulas da lógica de primeira ordem onde um literal é positivo, assim obtêm-se regras o formato *cabeça* ← *corpo*, onde *cabeça* era o literal positivo e *corpo* é a conjunção dos literais que eram negados na disjunção. Assim a clásula de Horn $Pred_1(\mu_1)$ *v* ~ $Pred_2(\mu_2)$ *v ...* *v* ~$Pred_n(\mu_n)$ torna-se $Pred_1(\mu_1)$ ← $Pred_2(\mu_2)$,..,$Pred_n(\mu_n)$ onde a vírgula (,) denota *e*.

O exemplo: *avô (X,Y)* ← *pai_mae (X,Z), pai_mae(Z,Y)* é uma regra ILP que significa: uma pessoa X é avô de uma pessoa Y, se uma pessoa Z pode ser encontrada, tal que, X é pai ou mãe de Z e Z é pai ou mãe de Y. A parte do lado esquerdo de ← é denominada *cabeça* e a parte à direita é chamada *corpo* da cláusula. As vírgulas significam a conjunção e X,Y e Z são variáveis quantificadas universalmente. As palavras *avô*, *pai_mae* são chamadas de predicados e as variáveis entre parênteses são chamadas de argumentos [BRA98].

Um sistema de programação lógica indutiva pode ser dividido em várias dimensões [DZE96]:

1. O sistema pode aprender conceitos simples ou múltiplos (predicados);

2. Pode exigir que todos os exemplos de treinamento tenham sido dados antes do processo de aprendizado, i.e., aprendizado em lote (*batch learners*), ou pode aceitar exemplos um por um ,.i.e., aprendizado incremental (*incremental learners*);

3. Durante o processo de aprendizagem, o aprendiz (programa que aprende) pode confiar num oráculo[4] para verificar a validação de generalizações e/ou exemplos de classificação gerados pelo mesmo. Este processo de aprendizagem é chamado de interativo;

4. O aprendiz pode inventar novos termos (predicados), isto resulta na extensão do vocabulário que pode facilitar a tarefa de aprendizado; e

5. O aprendiz pode tentar aprender um conceito a partir de um conjunto vazio ou pode aceitar uma hipótese inicial (teoria) a qual é revisada no processo de aprendizado. Aqueles que aceitam uma hipótese inicial são denominados de revisores de teoria (*theory revisors*).

Embora as dimensões descritas são, em princípio, independentes, os sistemas ILP existentes são situados nas duas pontas do espectro. Em uma ponta estão sistemas que rodam em lote e não interativos que aprendem predicados simples a partir de um conjunto vazio, enquanto na outra ponta, são interativos, incrementais e revisores de teoria que aprendem a partir de predicados múltiplos [DZE96].

### 3.2.1 Mecanismo de Tendência

Tendência é qualquer coisa que influencia como o aprendiz de conceito gera as inferências indutivas baseado nas evidências, com exceção do critério de consistência com os dados. Existem, fundamentalmente, duas formas de tendências:

- Tendência declarativa: define o espaço de hipótese a ser considerado pelo aprendiz, i.e., o que procurar; e
- Tendência de preferência: determina como procurar no espaço de hipótese definido pela tendência declarativa, i.e., quais hipóteses levar em consideração, quais hipóteses devem ser modificadas, entre outros.

[4] pessoa cujo o conselho tem uma importância ou no qual podemos confiar totalmente

A tendência de preferência é utilizada quando existem várias hipóteses que descrevem todos os exemplos e é necessário alguma base para graduar essas hipóteses, ou seja, decidir qual(is) é(são) mais relevante(s).

A tendência declarativa é dividida em dois tipos: sintática e semântica. A tendência sintática impõe restrições na forma das cláusulas permitidas na hipótese, e.g., existir apenas cláusulas positivas na hipótese. A tendência semântica impõe restrições no significado, ou no comportamento das hipóteses. Para ilustrar a tendência de semântica pode-se citar os *tipos* e os *modos* de uma linguagem de hipótese, onde *tipos* pode significar quais os tipos de dados são aceitos nos argumentos das literais candidatas a hipóteses e *modos*, quais literais são candidatas a cabeça da hipótese e quais serão candidatas ao corpo.

A maioria dos sistemas ILP, tais como , FOIL[QUI90], Progol[MUG00], RDT/DB[BRO96], entre outros, utilizam os mecanismos de tendência para reduzir o espaço de busca e definir um conjunto de hipóteses/regras a serem descobertos.

### 3.2.2 Exemplo

Esta subseção apresenta um exemplo de como a máquina de inferência indutiva de ILP trabalha[5]. Este exemplo será utilizado ao longo deste trabalho [MUG94]. O objetivo é aprender os relacionamentos entre pessoas em uma família. É informado que avô é o pai de um de seus pais, mas não temos o que é *pai_mãe*. Se o seguinte conhecimento prévio do domínio B é dado:

*avô (X,Y)* ← *pai(X,Z), pai_mae(Z,Y).*

*pai(henry, jane)* ←.

*mae(jane,john)* ←.

*mae(jane, alice)* ←.

E os seguintes fatos que descrevem os avôs e seus netos (exemplos positivos E+):

*avô(henry, john)* ←.

*avô(henry, alice)* ←.

Os seguintes fatos  onde o relacionamento não atende (exemplos negativos E-):

---

[5] As cláusulas no formato *cabeça* ← indicam que a hipótese/regra é positiva/verdadeira, e no formato ← *corpo* indica falsa/negativa.

← *avô (john,henry).*

← *avô (alice, john).*

Portanto, acreditando em B, e face aos exemplos positivos e negativos, i.e., E+ e E-, pode-se concluir:

H = *pai_mae(X,Y)* ← *mae(X,Y).*

A hipótese H não é um conseqüencia de B e E-, isto é, B∧E- ⊭□. Esta propriedade é denominada de satisfação anterior.

A hipótese H permite explicar os exemplos positivos E+ relativos a B, ou seja, B∧H⊨ E+. Propriedade conhecida como suficiência posterior.

O conhecimento prévio do domínio B e a hipótese H são consistentes em relação aos exemplos negativos E-, tal que, B∧H∧E-⊭□. Conhecida como satisfação posterior.

Satisfação anterior, suficiência e satisfação posterior serão discutidos na próxima Seção.

## 3.3 Modelo da Teoria de ILP

Os elementos lógicos (a semântica) que envolve a inferência indutiva são apresentados nesta seção, juntamente com os relacionamentos que existem entre os mesmos. É descrito dois tipos de semânticas para ILP: a normal e a não-monotônica, além da semântica definida, que é um tipo especial da semântica normal. A diferença básica entre essas duas semânticas é que a primeira trabalha com exemplos positivos e negativos e a segunda apenas com positivos.

Na definição destes tipos de semânticas, será considerada a noção de tendência de sintaxe *(syntatic bias)*, que define o conjunto de hipóteses bem formuladas constituindo um parâmetro de entrada para qualquer tarefa de ILP.

### 3.3.1 Semântica Normal

Semântica normal utiliza uma configuração genérica de ILP e permite que exemplos, teorias adquiridas e hipóteses sejam qualquer forma lógica bem formada.

O problema da inferência indutiva é, dado um conhecimento prévio do domínio B e uma evidência E, i.e., conjunto de exemplos positivos e negativos, E=E+∧E-, encontrar uma hipótese H tal que as condições a seguir sejam satisfeitas [DZE96]:

1. Satisfação Anterior: B∧E-⊭□

2. Satisfação Posterior (consistência): B∧H∧E-⊭□

3. Necessidade Anterior: B⊭E+

4. Suficiência Posterior (completude): B∧H⊨ E+, os exemplos positivos são conseqüência lógica do conhecimento prévio do domínio e das hipóteses encontradas.

Em alguns casos a teoria e as hipóteses são restringidas para serem bem definidas. A abordagem definida é mais simples que a genérica pois a teoria de cláusulas definidas (T) tem um único modelo mínimo de Herbrand M+(T), e qualquer fórmula lógica é verdadeira ou falsa neste modelo mínimo. Esta definição é considerada como *semântica definida*, a mesma é definida segundo[MUG94] como:

1. Satisfação Anterior: todo o exemplo que pertença às evidências negativas é falso no modelo mínimo de Herbrand para o conhecimento prévio do domínio, i.e., ∀e ∈ E- é falso em M+(B).

2. Satisfação Posterior: todo o exemplo que pertença às evidências negativas é falso no modelo mínimo de Herbrand para o conhecimento prévio do domínio e a hipótese, i.e., ∀e ∈ E- é falso em M+(B∧H).

3. Necessidade Anterior: algum exemplo que pertença às evidências positivas é falso no modelo mínimo de Herbrand para o conhecimento prévio do domínio, i.e., ∃e ∈ E+ que é falso M+(B).

4. Suficiência Posterior: todo o exemplo que pertença às evidências positivas são verdadeiros no modelo mínimo de Herbrand para o conhecimento prévio do domínio e a hipótese, i.e., ∀e ∈ E+ são verdadeiros em M+(B∧H).

Para maiores detalhes sobre o modelo de Herbrand veja [RUS95].

Um caso especial de semântica definida, onde a evidência é restrita a fatos verdadeiros e falsos (exemplos), é chamado de configuração por exemplos. Equivalente à semântica normal onde B e H são cláusulas definidas e E é um conjunto de cláusulas valoradas. Cláusulas de evidências

geralmente capturam mais conhecimento que os fatos de evidência considerando apenas os fatos valorados. Se *avô (henry,john)* ← *pai (henry, jane), mae (jane, john)*, uma evidência positiva poderia ser ←*avô (X,X)*, que indica que ninguém pode ser avô dele mesmo [MUG94]. A evidência positiva, na realidade, se comporta como um conceito que o sistema descarta durante o processo de aprendizado, evitando processar esses tipos de hipóteses.

A utilização da evidência por cláusula oferece uma especificação incompleta ou parcial dos predicados. Isto restringe o espaço de hipóteses aceitáveis. Evidência positiva tem que ser verdadeira no modelo mínimo de hipótese e teoria, e a negativa deve ser falsa.

### 3.3.2 Semântica Não-Monotônica

A teoria prévia do domínio – *background theory* - é um conjunto de cláusulas definidas. A evidência é vazia pois as evidências positivas são consideradas como parte da teoria prévia do domínio e as evidências negativas são baseadas na hipótese do mundo fechado. As hipóteses são um conjunto genérico de cláusulas que são expressas usando o mesmo alfabeto da teoria prévia do domínio.

Na semântica não-monotônica, as seguintes condições devem valer para H e B [DZE96]:

1. Validade: todas as hipóteses que pertençam ao conjunto das hipóteses são verdadeiras no modelo mínimo de Herbrand, i.e., $\forall h \in H$ é verdadeiro no M+(B)
2. Completude: Se uma determinada cláusula genérica *g* é verdadeira em M+(B) então $H \models g$.
3. Minimalidade: Não existe um subconjunto próprio G de H o qual é válido e completo (itens 1 e 2).

A validade garante que todas as cláusulas que pertençam à hipótese são verdadeiras no banco de dados B, ou seja, são propriedades verdadeiras dos dados. A completude garante que toda a informação válida no banco de dados deve ser codificada na hipótese. Esta exigência deve ser dada como tendência de sintaxe (*sintax bias*) a qual determina o conjunto de hipóteses bem formadas. A minimalidade garante a derivação de hipóteses não redundantes.

A semântica não-monotônica realiza a indução por dedução: uma hipótese H, deduzida da observação de exemplos E e teoria de domínio B, é verdadeira para todos os conjuntos possíveis de exemplos. Isto produz a generalização além da observação. Propriedades derivadas do conjunto

não-monotônico são mais conservadoras que aquelas derivadas do conjunto normal. A diferença entre ambas está no fato da utilização da hipótese do mundo fechado.

Qualquer hipótese H com suficiência posterior e satisfação posterior para uma teoria adquirida B e exemplo E tal que E-=M-(B∧E+) é válido no conjunto não-monotônico se Bp= B (B∧H) = B (B∧E+), onde Bp é a base de Herbrand [MUG94].

### 3.4 Estruturas Básicas para Inferência de Regras

As estruturas para inferências de regras em ILP procuram por cláusulas na linguagem de hipótese que são consistentes com os exemplos de treinamento. O espaço das cláusulas é estruturado pela relação de generalização entre as mesmas, chamado de θ-subjugamento. Uma cláusula *c* é mais genérica que uma cláusula *d* se *c* subjuga *d*. Por exemplo: a cláusula *filha (X,Y)* ← *feminina (X)* é mais geral que a cláusula *filha (ana, Y)* ← *feminina (ana), pai_mae (Y, ana)* [DZE96].

A seguir serão apresentadas algumas estruturas para inferências de regras para ILP que são relevantes ao processo de KDD, o qual é objeto deste trabalho: a generalização mínima relativa, resolução inversa, procura em grafo de refinamento, utilização de modelo de regras e transformação de problemas de ILP na forma proposicional [DZE96] e *Inverse Entailment* [MUG95]. Os exemplos apresentados utilizam os dados apresentados na Tabela 3.2.

Tabela 3.2 – Exemplo de Relacionamento *Filha*

| **Exemplos de Treinamento** | **Conhecimento Prévio do Domínio** | | | |
|---|---|---|---|---|
| filha(sue,eva). (positivo) | mae(eva,sue). | pai_mae(X,Y)← mae(X,Y). | feminina(ana). | masculino(pat). |
| filha (ana,pat). (positivo) | mae(ana,tom). | pai_mae(X,Y)← pai(X,Y). | feminina(sue). | masculino(tom). |
| filha(tom,ana). (negativo) | pai(pat,ana). | | feminina(eva). | |
| filha(eva,ana). (negativo) | pai(tom,sue). | | | |

#### 3.4.1 Generalização Mínima Relativa

A noção de generalização mínima[6] (lgg) é importante para ILP pois a mesma forma a base para a generalização cautelosa. Esta estrutura assume que se duas cláusulas, $c_1$ e $c_2$, são verdadeiras, provavelmente a generalização mínima *lgg*($c_1$, $c_2$) das mesmas também o será. O *lgg* de duas

[6] least general generalization

cláusulas é computado através do *lgg* de cada par de literal no cabeça e no corpo da cláusula. O mesmo é calculado substituindo as partes que não combinam por variáveis, e.g., se $c_1$ = *filha(maria, ana)* ← *feminina (maria), pai_mae(ana, maria)* e $c_2$ = *filha(eva, tom)* ← *feminina (eva), pai_mae(tom,eva),* então o *lgg*($c_1$, $c_2$) = *filha(X,Y)* ← *feminina (X), pai_mae (Y,X),* onde X indica o *lgg (maria, eva)* e o Y o *lgg (ana,tom).*

A generalização mínima relativa[7] (*rlgg*) de duas cláusulas, $c_1$ e $c_2$, é a cláusula menos genérica que é mais genérica que $c_1$ e $c_2$ com respeito (em relação) ao conhecimento prévio do domínio B. Esta técnica é utilizada em sistemas onde o conhecimento prévio do domínio é restrito aos fatos valorados. Se K é uma conjunção de todos estes fatos, o *rlgg* de dois átomos valorados $A_1$ e $A_2$ (exemplos positivos), relativos à *B*, é definido como [DZE96]:

$$rlgg(A_1, A_2)=lgg((A_1 \leftarrow K), (A_2 \leftarrow K))$$

Dado exemplos positivos $e_1$=*filha(maria,ana)* e $e_2$=*filha(eva,tom),* e um conhecimento prévio do domínio *B* (Tabela 3.2). A generalização mínima relativa de $e_1$ e $e_2$ relativa a *B* é calculada como:

$$rlgg(e_1, e_2)=lgg(e_1 \leftarrow K), (e_2 \leftarrow K)$$

Onde K é a conjunção das literais *pai_mae(ana,maria), pai_mae(ana,tom), pai_mae(tom,eva), pai_mae(tom,ian), feminina(ana), feminina(maria)* e *feminina(eva).* Para conveniência será adotadas seguintes abreviações: *d-daughter, p-pai_mae, f-feminina, a-ana, e-eva, m-maria, t-tom, i-ian*. A conjunção dos fatos a partir do conhecimento prévio do domínio é:

$$K=p(a,m), p(a,t),p(t,e),p(t,i),f(a),f(m),f(e)$$

O cálculo do $rlgg(e_1,e_2)$ produz a seguinte cláusula::

$$d(V_{m,e}, V_{a,t}) \leftarrow p(a.,m), p(a,t), p(t,e), p(t,i), f(a), f(m), f(e), p(a,V_{m,t}), p(V_{a,t},V_{m,e}), p(V_{a,t},V_{m,i}), p(V_{a,t},V_{t,e}), p(V_{a,t},V_{t,i}), f(V_{a,m}), f(V_{a,e}), f(V_{m,e}).$$

Onde $Vx,y$ representa $rlgg(x,y)$ para cada *x* e *y*. Genericamente, um *rlgg* de um conjunto de exemplos de treinamento pode conter literais infinitas ou, no mínimo, crescer exponencialmente com o número de exemplos. Para evitar esse problema alguns sistemas usam algum tipo de restrição para a introdução de novas variáveis no corpo dos *rlgg*. Mesmo utilizando algumas restrições, *rlggs* são geralmente cláusulas longas cheias de literais irrelevantes.

---

[7] relative least general generalization

Eliminado as literais irrelevantes do cálculo do *rlgg* anterior tem-se:

$d(V_{m,e}, V_{a,t}) \leftarrow p(V_{a,t}, V_{m,e}), f(V_{m,e})$.

Que é representada por *filha(X,Y)←feminina(X), pai_mae(Y,X)*.

### 3.4.2 Resolução Inversa

As regras de referência indutiva podem ser vistas como o inverso das regras dedutivas de inferência. Desde que um regra dedutiva de resolução é completa para dedução, uma inversão da resolução deveria ser completa para indução. Esta é a idéia básica da resolução inversa[MUG94].

O passo básico de resolução na lógica proposicional deriva da proposição *p v r* dadas as premissas *p v~q* e *q v r*. Na lógica de primeira ordem, a resolução é mais complicada, envolvendo substituições. A conclusão alcançada das cláusulas *c* e *d* através do passo da resolução de inferência é denotada por *res (c,d)* e é chamada de resolvente de *c* e *d* [DZE96]

A resolução inversa de primeira ordem será ilustrada utilizando o seguinte estudo de caso:

*B* { $b_1$ *= feminina (maria)* e $b_2$*= pai_mae(ana,maria)*}

*H* {{c}={*filha (X,Y)* ← *feminina (X), pai_mae (Y,X)*}

T=*H* U *B*

Supondo que se queira derivar o fato *filha (maria,ana)* de T, então deve-se proceder:

1. Primeiro, o resolvente $c_1$=res(c, $b_1$) é calculado sob a substituição $\theta_1$ ={X/maria}. Para tal aplica-se a substituição em *c* para obter *filha(maria, Y)* ← *feminina (maria), pai_mae(Y,maria)*, o qual é, então, resolvido com $b_1$ como no caso proposicional. O resolvente de *filha(X,Y)←feminina(X), pai_mae(Y,X)* e *feminina(maria)* é $c_1$*=res(c,*$b_1$*)=filha(maria,Y)←pai_mae(Y,maria).*
2. O próximo resolvente $c_2$*=res(*$c_1$*,* $b_2$*)* é calculado sob a substituição $\theta_2$= {Y/ana}. A cláusula *filha (maria, Y)* ← *pai_mae (Y, maria) e pai_mae(ana, maria)* resolve $c_2$*=res(*$c_1$*,* $b_2$*) = filha(maria,ana)*

A resolução inversa utiliza o operador de generalização baseado na substituição invertida. Dado uma cláusula W, uma substituição inversa $\theta^{-1}$de uma substituição $\theta$ é uma função que mapeia termos em W$\theta$ para variáveis, tal que, W$\theta\theta^{-1}$ =W. Seja *c = filha (X,Y)* ← *feminina (X), pai_mae (Y,X)* e a substituição $\theta$ = {X/maria, Y/ana}, *c' = c*$\theta$ *= filha (maria,ana)* ← *feminina (maria),*

*pai_mae(ana,maria)*, aplicando-se a resolução inversa, $\theta^{-1}$={maria/X, ana/Y}, a cláusula original *c* é obtida.

Em casos genéricos, a substituição inversa é, substancialmente, mais complexa pois envolve a colocação de termos para garantir que as variáveis na cláusula inicial W sejam restauradas, apropriadamente, em $W\theta\theta^{-1}$.

Baseado no caso apresentado anteriormente, apenas considerando que H= Ø, e que o aprendiz encontrou o exemplo positivo $e_1$=*filha (maria,ana)*. O processo de resolução inversa procede como se segue:

1. No primeiro passo, a técnica de resolução inversa tenta encontrar uma cláusula $c_1$ que, junto com $b_2$, implica $e_1$ e pode ser adicionada para a hipótese corrente, ao invés de $e_1$. Utilizando a substituição inversa $\theta_2^{-1}$={ana/Y}, este passo gera a cláusula $c_1$=*ires*($b_2$,$e_1$)=*filha(maria,Y)* ← *pai_mae(Y, maria)*. A cláusula $c_1$ torna-se a hipótese corrente, tal que {$b_2$}UH |- $e_1$.

2. Então, no segundo passo define-se $b_1$=*feminina* (maria) e a hipótese corrente H={$c_1$} = {*filha(maria,Y)* ← *pai_mae(Y,maria)*}. Computando *c' = ires* ($b_1$, $e_1$), usando a substituição inversa $\theta_1^{-1}$={maria/X}, é generalizada a cláusula $c_1$ em relação ao conhecimento prévio do domínio B, levando *c' = filha (X,Y)* ← *feminina (X), pai_mae (Y,X)*. Na hipótese corrente, a cláusula $c_1$ pode ser trocada pela cláusula mais geral *c'*, a qual, junto com B implica o exemplo $e_1$. A hipótese induzida, então, é *filha(X,Y)*←*feminina(X), pai_mae (Y,X)*.

O operador de generalização utilizado é chamado de operador de absorção ou operador em V. A Figura 3.3 apresenta a árvore gerada pelo mesmo. O operador identidade também é um operador em V. Outros operadores importantes são de intra-construção e inter-construção, conhecidos como operador em W, o qual combina dois operadores em V [MUG94]:

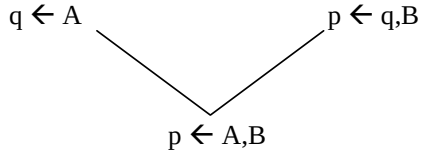
1. Absorção: $\underline{q \leftarrow A \qquad p \leftarrow A,B}$

   $q \leftarrow A \qquad p \leftarrow q,B$

2. Identificação: $\underline{p \leftarrow A,B \qquad p \leftarrow A,q}$

   $q \leftarrow B \qquad p \leftarrow A,q$

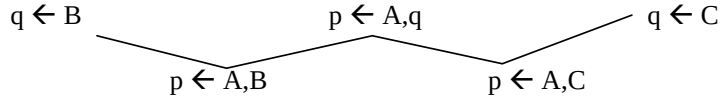
Representação em V:




3. Intra-Construção: $\frac{p \leftarrow A,B \qquad\qquad p \leftarrow A,C}{q \leftarrow B \qquad p \leftarrow A,q \qquad q \leftarrow C}$

4. Inter-Construção: $\frac{p \leftarrow A,B \qquad\qquad q \leftarrow A,C}{p \leftarrow r,B \qquad r \leftarrow A \qquad q \leftarrow r,C}$

Representação em W:




Os operadores em W geram cláusulas usando predicados não disponíveis no vocabulário inicial do aprendiz. A habilidade de introduzir novos predicados é conhecida como *invenção de predicados*.

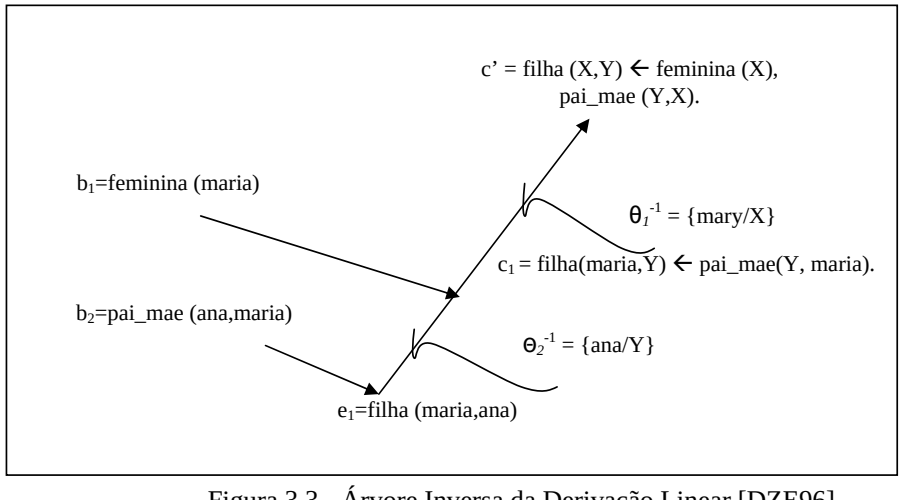



Figura 3.3 - Árvore Inversa da Derivação Linear [DZE96]

### 3.4.3 Busca em Grafo de Refinamento

A técnica básica de especialização em ILP é busca *top-down* de grafos refinados. O aprendiz *top-down* inicia da cláusula mais genérica e refina, repetidamente, até a mesma não cobrir mais exemplos negativos. Este refinamento pode ser entendido como especialização. Durante a busca é garantido que a cláusula cobre, pelo menos, um exemplo positivo [DZE96].

Esta técnica funciona da seguinte maneira: para uma linguagem de hipóteses *L* e um conhecimento prévio do domínio B, o espaço de hipóteses de cláusulas de programa é um reticulado, i.e., um conjunto de cláusulas reduzidas, estruturado pela generalização ordenada do θ-subjugamento. Neste reticulado, um grafo de refinamento pode ser definido como grafo direto, acíclico, onde nodos são cláusulas de programa e arcos correspondem às operações básicas de refinamento: substituição de uma variável por um termo e adição de um literal ao corpo de uma cláusula.

Parte do grafo de refinamento do problema do exemplo anterior é apresentado na Figura 3.4. A busca inicia com a cláusula *filha (X,Y)* ←, a qual cobre dois exemplos negativos (Tabela 3.2). Os refinamentos desta cláusula são processados, dos quais *filha (X,Y)* ← *feminina (X)* e *filha (X,Y)* ← *pai_mae (Y,X)*, cobrem apenas um exemplo negativo (Tabela 3.2). Estas duas cláusulas têm um refinamento comum: *filha(X,Y)*←*feminina (X),pai_mae(Y,X)*, o qual não cobre nenhum exemplo negativo [DZE].

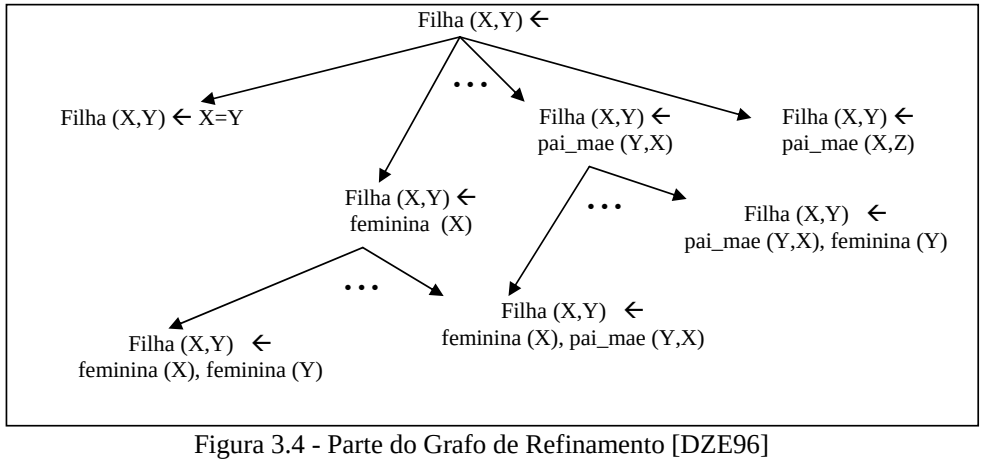



Figura 3.4 - Parte do Grafo de Refinamento [DZE96]

### 3.4.4 Utilização de Modelo de Regras

Modelos de regras são uma forma de influência de linguagem declarativa, i.e., qualquer descrição do espaço de hipótese de possíveis programas para o sistema aprender[BER96]. Estes

modelos explicitamente especificam a forma das cláusulas que podem aparecer nas hipóteses e são usados como *templates* para construir as mesmas [DZE96]. Estes *templates* são oferecidos pelos usuários ou podem ser derivados de regras aprendidas anteriormente através de uma ferramenta de aquisições de modelos, por exemplo.

Um modelo de regra R tem a forma $T \leftarrow L_1,...,L_m$, onde T e $L_i$ são esquemas de literais. Cada esquema $L_i$ tem a forma $Q_i$ $(Y_1,...,Y_{ni})$ ou $\sim Q_j$ $(Y_1,...,Y_{nj})$ onde todas as variáveis não-predicados $Y_k$ são implicitamente universalmente quantificadas. A variável predicado $Q_i$ pode ser instanciada por apenas um símbolo de predicado de grau $n_i$.

Para o aprendizado da relação *filha* do exemplo anterior, o usuário necessita definir o seguinte modelo: $P(X,Y) \leftarrow R(X), Q(X,Y)$, neste caso, a cláusula correta *filha (X,Y)* ← *feminina (X), pai_mae (Y,X)* pode ser induzida através de substituições das literais do modelo por literais do conhecimento prévio do domínio que se encaixem na substituição. O modelo sugerido poderia ser instanciado para a cláusula *filho* (X,Y) ← *masculino* (X), *pai_mae* (Y,X). Portanto, um modelo de regra pode ser útil para resolver vários problemas de aprendizado.

O espaço de cláusulas especificadas pelo modelo de regras é muito grande. Para cada modelo, todas as possíveis instanciações de variáveis de predicados são tentadas sistematicamente e o resultado é testado em relação aos exemplos e ao conhecimento prévio do domínio. Uma ordem hierárquica dos modelos de regras, baseada numa extensão do θ-subjugamento, pode ajudar algumas implementações de ILP a podar o espaço de hipóteses.

### 3.4.5 Transformação de Problemas de ILP na Forma Proposicional

Esta técnica baseia-se na transformação da forma relacional de um problema ILP para a forma atributo-valor. Assim o problema é resolvido por um aprendiz de atributo-valor. Esta abordagem é própria apenas para uma classe restrita de problemas em ILP. A linguagem de hipóteses é restrita para cláusulas de programas livres de funções que são: 1) tipadas – cada variável é associada com um determinado conjunto de valores; 2) restritas – todas as variáveis no corpo da cláusula também aparecem na cabeça, e 3) não-recursivas – símbolos de predicados que aparecem na cabeça, não podem aparecer no corpo.

O primeiro passo de um algoritmo que utiliza esta técnica é determinar as possíveis aplicações dos predicados adquiridos nos argumentos das relações destino, levando em conta os tipos de argumentos. Cada uma dessas aplicações introduz um novo atributo. Baseado no exemplo da relação de família, o aprendiz de atributo-valor correspondente é apresentado na Tabela 3.3,

onde f é *feminina*, m é *masculino* e p é *pai_mae*. As tuplas de atributo-valor são generalizações – relativa ao conhecimento prévio do domínio – dos fatos individuais sobre a relação destino. Nesta tabela, as variáveis são os argumentos daquela relação, e as características proposicionais denotam os atributos mais recentes construídos pela tarefa de aprendizado proposicional. No aprendizado de cláusulas livres de funções, apenas os novos atributos são considerados para o aprendizado.

No segundo passo, o programa de aprendizado de atributo-valor induz a seguinte regra *if-then* a partir das tuplas da tabela apresentada:

*class= (exemplo positivo) if [feminina (X) = true ] ∧ [pai_mae(Y,X) = true]*

No último passo, as regras *if-then* são transformadas em cláusulas de primeira ordem. No exemplo, *filha (X,Y) ← feminina (X), pai_mae (Y,X).*

Tabela 3.3 - Forma Proposicional do Problema da Relação da Família [DZE96]

| classe | Variáveis | | Características proposicionais | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | X | Y | f(X) | f(Y) | m(X) | m(Y) | p(X,X) | p(X,Y) | p(Y,X) | p(Y,Y) |
| + | maria | ana | V | V | F | F | F | F | T | F |
| + | Eva | tom | V | F | F | T | F | F | T | F |
| - | Tom | ana | F | V | T | F | F | F | T | F |
| - | Eva | Ana | V | V | F | F | F | F | F | F |

### 3.4.6 *Inverse Entailment*

*Inverse Entailment* é uma regra de inferência indutiva proposta por Muggleton[MUG95]. Este método é completo e eficiente para inverter implicação entre cláusulas livres de função.

O problema genérico de especificação de ILP, como apresentado anteriormente, é: dado um conhecimento prévio do domínio B e exemplos E, encontrar a hipótese H mais simples e consistente (onde simplicidade é uma medida relativa à distribuição *a priori*) tal que $B \wedge H \models E$.

Em geral, B, H e E podem ser programas lógicos arbitrários. Cada cláusula no mais simples H deverá cobrir, no mínimo, um exemplo, senão existirá um H' mais simples que cobrirá. Considera-se, então, o caso de H e E serem cláusulas simples de Horn. Isto pode ser visto agora como uma forma generalizada de absorção para gerar $B \wedge H \models E$. Se rearranjar este programa utilizando a lei da contraposição, consegue-se uma forma mais simples $B \wedge \sim E \models \sim H$. Como H e E são cláusulas simples de Horn, a negação delas gerará programas lógicos consistindo apenas de

unidades cláusulas instanciadas. Considerando ~⊥ uma conjunção de literais instanciadas (potencialmente infinita), as quais são verdadeiras em todos os modelos de B∧ ~E. Como ~H é verdadeiro para todos os modelos de B∧~E, conterá todos os subconjuntos de literais instanciadas em ~⊥. Portanto B∧~E⊨~⊥⊨~H, assim para todo H tal que H⊨⊥, um subconjunto de soluções para H pode ser encontrado considerando as cláusulas que θ-subjugam ⊥. O conjunto completo de candidatos para H pode ser encontrando considerando todas as cláusulas que θ-subjugam ⊥ [MUG00]. Utilizando-se, assim, o princípio da *Inverse Entailment* para induzir regras.

### 3.5 ILP e Mineração em Banco de Dados

ILP tem sido empregada de várias maneiras para minerar dados em banco de dados [BLO96,BRO96,DZE96,MUG97]. S. Dzeroski em [DZE96] afirma que ILP pode ser utilizado para descobrir padrões envolvendo várias relações em banco de dados, pois o poder expressivo de uma linguagem de padrões é uma forte motivação para uso de ILP no contexto de descoberta do conhecimento em banco de dados (KDD).

P. Brockhausen e K. Morik em [BRO96] afirmam que algoritmos de aprendizado de regras do tipo ILP são de particular interesse para KDD devido ao fato de permitirem a detecção de regras mais complexas que em outros tipos de algoritmos. Porém existem poucos algoritmos baseados em ILP que acessem bancos de dados. Como a demanda para KDD é analisar bancos de dados para a descoberta do conhecimento, muitos autores de sistemas ILP estão alterando seus sistemas para atender a essa necessidade, e.g., RDT possui uma versão para este fim, o RDT/DB[BRO96].

### 3.6 Áreas de Pesquisa

As áreas de pesquisas em ILP são bastante extensas: 1) aprendizado de predicado múltiplos, 2) uso da indução para determinar restrições de integridade válidas ou quase válidas em banco de dados[DZE96], 3) a necessidade de sistemas ILP trabalharem eficientemente e efetivamente com dados numéricos e 4) aprendizado de teorias de cláusulas completas[MUG94], etc.

### 3.7 Considerações Finais

A Programação Lógica Indutiva pode ser vista como aprendizado de máquina em uma linguagem de lógica de primeira ordem, onde as relações podem ser apresentadas no contexto de banco de dados dedutíveis. Isso é relevante para o descoberta do conhecimento em bancos de dados relacionais, podendo descrever padrões que envolvam mais de uma relação.

Assim ILP é considerada uma das técnicas mais eficazes para a descoberta de padrões pois a mesma oferece uma forma poderosa para representar restrições, gramáticas, planos, equações, e relacionamentos temporais e espaciais. As linguagens de programação lógica, tal como Prolog, usam um sub-conjunto da lógica matemática para oferecer um linguagem rica para representação declarativa. Enquanto a maioria das técnicas de aprendizado de máquina fazem a aquisição do conhecimento de modo procedural, ILP oferece uma forma declarativa de aquisição utilizando exemplos, conhecimento prévio do domínio e hipóteses [MUG97].

Este capítulo descreveu a maioria das técnicas utilizadas por um sistema ILP. O próximo capítulo discutirá alguns sistemas ILP estudados que implementam as técnicas descritas. Esse estudo possibilitou o desenvolvimento do sistema descrito nesse trabalho (DBILP).

# Capítulo 4

## 4 Sistemas ILP Estudados

Este capítulo apresenta alguns sistemas ILP estudados para possibilitar o desenvolvimento do DBILP. Primeiramente estudos teóricos foram realizados, identificando quais técnicas os sistemas utilizam para resolver o problema de ILP. Em seguida, para alguns sistemas, foi realizado um estudo empírico. Nem todos os sistemas puderam ser estudados empiricamente pois não possuem versões de domínio público ou a plataforma onde executam não está disponível no ambiente onde esse trabalho foi desenvolvido.

### 4.1 Progol

Progol combina *inverse entailment* com a busca da cláusula mais genérica para a mais específica através de um grafo de refinamento. *Inverse entailment* é usado com as declarações *modes* para derivar a cláusula mais específica, a qual cobre um dado exemplo. Esta cláusula é usada para guiar a busca no grafo de refinamento. Diferente dos métodos de busca utilizados em outros sistemas, e.g., MIS [SHA93], FOIL [QUI90], a busca utilizada em Progol é eficiente e tem uma garantia provável de retorno de uma solução tendo o máximo de compressão do espaço de busca [MUG95]. Para fazer isso, é executado um método de busca semelhante ao A* [RUS95], guiado pela compressão, sobre cláusulas que subjugam a cláusula mais específica. Progol é um procedimento de aprendizado de máquina e uma forma de programação lógica indutiva [ROB97].

Progol possui um linguagem de entrada e esta linguagem possuem uma declaração *modes* onde o usuário declaradas as cláusulas candidatas à cabeça e ao corpo das hipóteses, bem como seus os argumentos. Nesta linguagem também é declarado o tipo de cada argumento. Além das declarações citadas, que implementam as tendências de preferência e declarativa, a linguagem de entrada do Progol permite que seja declarado os exemplos positivos e negativos e o conhecimento prévio do domínio. A Figura 4.1 apresenta um programa Progol utilizado como base para descrever como Progol constrói a cláusula mais específica[MUG00].

```
%% Modes
%% attempt to teach implication in a 5-valued logic
:-modeh(1,implies5(+truthvalue,+truthvalue, -truthvalue))?
:-modeb(1,or5(+truthvalue,+truthvalue, -truthvalue))?
:-modeb(1,not5(+truthvalue,-truthvalue))?
%%Types
truthvalue(0).
truthvalue(1).
truthvalue(2).
truthvalue(3).
truthvalue(4).
%% Background Knowledge
not5(X,Y) :- Y is 4-X.
or5(X,X,X).
or5(X,Y,Z):-X>Y, Z is X.
or5(X,Y,Z):-X<Y, Z is Y.
%% Positive examples (bodyless clause)
implies5(4,4,4).
implies5(4,0,0).
implies5(0,4,4).
implies5(0,0,4).
implies5(1,2,3).
%% Negative examples (headless clause)
:-implies5(2,0,0).
:-implies5(4,2,4).
```

Figura 4.1 - Programa Progol [MUG00]

Um programa *Progol* possui a seguinte sintaxe básica:

- A seção *modes* permite declarar o conceito que se está procurando (*modeh*) e as cláusulas que podem ser usadas neste conceito;
- Os argumentos das cláusulas podem ser do tipo +, - ou #, que indicam respectivamente, váriáveis de entrada, de saída e constantes;
- Todos os argumentos são tipos que devem ser valorados;
- Os exemplos positivos são regras/hipóteses sem corpo;
- Os exemplos negativos são regras/hipóteses sem cabeça; e
- O conhecimento prévio do domínio são cláusulas que definem os *modeh* declarados.

Os passos executados pelo *Progol* com o programa da Figura 4.1 são apresentados a seguir:

1 Escolhe: *implies5(4,4,4)* – primeiro exemplo positivo;

2 De *modeh(1,implies5(+truthvalue, +truthvalue, -truthvalue))* tem-se a dedução trivial: $B \wedge \sim e \models \sim implies5(4,4,4)$;

3 A partir de *modeb(1,or5(+truthvalue, +truthvalue, -truthvalue))*, substituindo-se as duas variáveis de entrada (variáveis precedidas por +) por 4 tem-se a dedução: $B \wedge \sim e \models or5(4,4,4)$;

4 Da declaração *modeb(1,not5(+truthvalue,-truthvalue))*, substituindo-se a variável de entrada por 4, tem-se a dedução: $B \wedge \sim e \models not5(4,0)$;

5 Agora pode-se usar tanto 4 como 0 como valores de entrada, assim tem-se as seguintes deduções também: $B \wedge \sim e \models or5(4,0,4)$, $B \wedge \sim e \models or5(0,4,4)$, $B \wedge \sim e \models or5(0,0,0)$ e $B \wedge \sim e \models not5(0,4)$; e

6 Não existe nenhum termo novo, assim estes são os últimos átomos instâncias para serem incluídos em $\sim\perp$. Unindo todos, tem-se: $B \wedge \sim e \models \sim implies5(4,4,4) \wedge or5(4,4,4) \wedge not5(4,0) \wedge or5(4,0,4) \wedge or5(0,4,4) \wedge or5(0,0,0) \wedge not5(0,4)$;

$\sim\perp$ é o lado direito da dedução. Um modelo mínimo de Herbrand é construído para $B \wedge \sim E$, a declaração de *mode* é utilizada para guiar a inclusão de predicados que podem ser importantes. Para derivar $\perp$, a dedução anterior é primeiramente negada para gerar:

$implies5(4,4,4) \vee \sim or5(4,4,4) \vee \sim not5(4,0) \vee \sim or5(4,0,4) \vee \sim or5(0,4,4) \vee \sim or5(0,0,0) \vee \sim not5(0,4)$

Assim a cláusula mais específica pode ser construída trocando os termos nesta dedução por variáveis únicas, gerando:

$\perp = implies5(A,A,A) \vee \sim or5(A,A,A) \vee \sim not5(A,B) \vee \sim or5(A,B,A) \vee \sim or5(B,A,A) \vee \sim or5(B,B,B) \vee \sim not5(B,A)$

E $\perp$ é dado por:

*implies5(A,A,A) :- or5(A,A,A), not5(A,B), or5(A,B,A), or5(B,A,A), or5(B,B,B), not5(B,A)*

na forma de cláusula.

O sistema Progol possui um conjunto de comandos em sua linguagem que permitem que a mesma seja configurada para trabalhar apenas com exemplos positivos, número máximo de átomos que uma hipótese construída possa ter, número máximo de nodos que serão visitados para a construção da hipótese geral, entre outros.

### 4.2 RDT/DB

RDT/DB é um sistema ILP definido por P. Brockhausen e K. Morik[BRO96] que possui uma interação entre a ferramenta de aprendizado e o gerenciador de banco de dados Oracle®, versão 7. O dicionário do banco de dados é utilizado para identificar as relações, atributos e tipos de dados, além das chaves para as tabelas do banco de dados. Estas tabelas são utilizadas para mapear relações e atributos em predicados da linguagem de hipóteses do RDT/DB. A geração de hipóteses é executada pela ferramenta de aprendizado, instanciando o esquema de regras e utilizando o algoritmo *breadth-first*[GRA96]. Para testar as hipóteses, consultas SQL são geradas pela ferramenta de aprendizado, sendo enviadas para o sistema de banco de dados. A interação entre a ferramenta de aprendizado e Oracle é feita através de conexão TCP/IP, a interface entre Prolog, a linguagem onde RDT/DB é implementada, e Oracle é escrita em C.

RDT/DB utiliza a mesma especificação declarativa para a linguagem de hipótese utilizada pelo seu antecessor o RDT[KIE92] e pela maioria dos sistemas ILP, para restringir o espaço de hipóteses. A especificação é dada pelo usuário em termos de esquema de regras. Um esquema de regras é uma regra com variáveis de predicado, ao invés de símbolos de predicado. Os argumentos das literais podem ser definidos por valores constantes de aprendizado. Como exemplos pode-se definir:

*mp_two_c(P1,P2,P3,C): P1(X1,C) & P2(Y,X1) & P3(Y,X2) → P1(X2,C).*

Neste exemplo, o segundo argumento da conclusão e o segundo argumento da primeira premissa literal são valores constantes que deverão ser aprendidos.

Para a geração das hipóteses, RDT/DB instancia as variáveis dos predicados e os argumentos que estão marcados para serem constantes para aprendizado. Um esquema de regra completamente instanciado é uma regra, e.g., *region(X1,europe) & licensed(Y,X1) & produced(Y,X2) → region(X2,europe)*, ou seja, todos os carros que são licenciados dentro da Europa também foram produzidos dentro da Europa.

O sistema RDT/DB utiliza as medidas de suporte e confiança para a validação das hipóteses, para tal, são contadas as tuplas que dão suporte a hipótese e as tuplas que a contradizem. A Figura

4.2 apresenta os comandos SQL que foram gerados para contar as tuplas suporte e contraditórias, baseado no exemplo apresentado. Pode-se observar nesses comandos SQL gerados que bastaria apenas um comando para contar os exemplos positivos e negativos. Este comando seria:

```
Select reg2.region, Count(*)
From   vehicles veh1, vehicles veh2,
       regions reg1, regions reg2
Where  reg1.place=veh1.produced_at
And    veh1.ID = veh2.ID
And    veh2.licensed=reg2.place
And    reg1.region='europe'
Group By  reg2.region;
```

Assim bastaria verificar o conteúdo de *reg2.region*, se fosse *europe* conta como exemplo positivo, caso contrário, negativo.

| Exemplos Positivos | Exemplos Negativos |
|---|---|
| Select Count(*)<br>From vehicles veh1, vehicles veh2,<br>regions reg1, regions reg2<br>Where reg1.place=veh1.produced_at<br>And veh1.ID = veh2.ID<br>And veh2.licensed=reg2.place<br>And reg1.region='europe'<br>And reg2.region='europe'; | Select Count(*)<br>From vehicles veh1, vehicles veh2,<br>regions reg1, regions reg2<br>Where reg1.place=veh1.produced_at<br>And veh1.ID = veh2.ID<br>And veh2.licensed=reg2.place<br>And reg1.region='europe'<br>And not reg2.region='europe'; |

Figura 4.2 - Comandos SQL Gerados pelo RDT/DB

### 4.3 FILP

O sistema FILP[BER96] resolve problemas através de métodos de cima para baixo (*top-down*) através de consultas feitas ao usuário para qualquer exemplos que esteja faltando, dependendo do espaço de hipótese que foi definido. Isto faz com que cada programa aprendido se comporte de forma exigida para o número de exemplos fornecidos, e um programa sempre é encontrado, caso exista. Também as hipóteses induzidas são restritas à programas lógicos que são funcionais, ou seja, aqueles que cada predicado de *n*-graus pode ser associado com funções da seguinte forma:

- *m* do seus argumentos são rotulados como de entrada;
- O restante dos argumentos (*n-m)* são rotulados como de saída;
- Para cada seqüência de valores de entrada fornecidos, existe apenas uma seqüência de valores de saída que faz o predicado ser verdadeiro e

- Os valores de saída devem sempre existirem e serem únicos, o programa deve corresponder com a função como um todo.

O espaço *C* de possíveis cláusulas é definido com uma linguagem de conjunto de cláusulas. As restrições funcionais citadas anteriormente são construídas no topo dos modos de entrada e saída que estão disponíveis no conjunto de cláusulas. Portanto, a funcionalidade deve ser informada pelo usuário e também é assumida a mesma funcionalidade para os exemplos não vistos.

As restrições citadas não afetam o poder de expressão. Como qualquer função computável pode ser representada por um programa lógico funcional, a atividade de aprendizado se torna mais fácil, pois muitas cláusulas que deveriam ser usadas para testar os exemplos são, *a priori*, descartadas.

Os predicados podem ser definidos intencionalmente, por meios de regras lógicas, ou extensionalmente, através da inserção de alguns exemplos para o seu comportamento de entrada e saída. Este sistema aceita cláusulas recursivas.

O funcionamento de FILP segue os seguintes passos:

- Gera a primeira cláusula. Considera o antecedente $\alpha$ vazio;
- Escolhe o primeiro literal para ser adicionado a $\alpha$ e
- Variáveis são retiradas da cláusula cabeça ou de um conjunto finito de variáveis adicionais tipadas.

Será apresentado um exemplo para explicar o funcionamento do sistema FILP. Neste exemplo FILP tenta aprender o conceito *reverse*, que retorna verdadeiro se o segundo termo for uma lista inversa do primeiro. Os predicados disponíveis são: *null*, *head. tail. reverse*, *assign* e *append*.

- Tendo $\alpha$*=null(Y)*. Um exemplo pode ser coberto mas a cláusula *reverse(X,Y):-null(Y)* não pode ser aceita pois devolve um número errado de saídas para valores de entrada, e.g., *reverse([a],[])*;
- Adicionando outra literal, tendo $\alpha$*=null(Y),head(X,H)*, neste caso nenhum exemplo positivo é coberto;
- Trocando a literal adicionada por $\alpha$*=null(Y),null(X)*, o exemplo *reverse([],[])* é coberto porém outros não, e.g., *reverse([a,b],[b,a]), reverse([a],[a])* e *reverse([a,b,c],[c,b,a])*;

- Combinando outras literais disponíveis, seria descoberto, finalmente, $\alpha$=*head(X,H), tail(X,T), reverse(T,W),append(W,[H],Y) e*
- Sendo todos os exemplos cobertos, teríamos as seguintes soluções: *reverse(X,Y):- null(X),null(Y)* e *reverse(X,Y):- head(X,H), tail(X,T), reverse(T,W),append(W,[H],Y).*

O sistema FILP é capaz de aprender de maneira eficiente programas lógicos clássicos usados em testes para sistemas ILP, e.g., *reverse, intersection, partition* e *quicksort.*

Os conceitos chave para o aprendizado do sistema FILP são:

- Sabe antecipadamente que está aprendendo uma função; somente necessitará de exemplos positivos do programa que precisa ser aprendido;
- Exige um número reduzido de exemplos positivos, principalmente pelo conhecimento que FILP tem sobre funções. Apenas sistemas baseados em implicação inversa seriam, em princípio, prontos para trabalhar com um número limitado de exemplos;
- FILP consulta o usuário em relação aos exemplos faltantes, assim o usuário não precisa informar todos os exemplos para FILP aprender um conceito pois o usuário será consultado quando faltar um determinado exemplo;
- FILP não aprende conceitos separadamente, ou seja, para aprender *quicksort* terá que ser aprendido junto o conceito de *partition.* O sistema pode aprender *quicksort* sem o conceito *partition* porém *quicksort* será aprendido apenas baseado nos exemplos definidos extensionalmente; e
- FILP aprende programas consistentes e completos.

### 4.4 GOLEM

GOLEM[MUG95,MUS97] é um sistema ILP que usa a generalização mínima relativa (*relative least general generalization - rlgg*) para generalizar até que todos os exemplos positivos sejam cobertos e nenhum exemplo negativo seja implicado. É usado um conhecimento prévio do domínio extensional para evitar o problema de procurar por *rlgg* infinitos.

O algoritmo do sistema GOLEM utiliza como entrada um conjunto de exemplos positivos $E+$ e um de exemplos negativos $E-$ para o conceito a ser aprendido, e um modelo de conhecimento prévio do domínio *B*, gerando como saída um conjunto de cláusulas que, junto com *B*, deriva todos os $E+$. Para gerar uma cláusula simples, GOLEM, primeiramente, escolhe aleatoriamente vários

pares de exemplos positivos, calcula os *rlgg*, e escolhe a cláusula com a maior cobertura. Esta cláusula é generalizada selecionando-se novos exemplos positivos aleatoriamente, e computando o *rlgg* da cláusula e de cada um dos exemplos. De novo, entres os *rlgg* resultantes o com maior cobertura é escolhido. O processo é repetido até que a cobertura de melhor cláusula pare de aumentar.

Para construir definições que consistem de mais de uma cláusula, GOLEM usa a abordagem de cobertura: constrói uma cláusula que cobre alguns exemplos positivos, remove os exemplos cobertos do conjunto de treinamento e repete todo o processo. Após a construção as cláusulas passam por um processo de ajuste. Este processo consiste de dois passos: primeiro é feita uma dedução lógica e então é executado o algoritmo de redução.

GOLEM pode aprender programas lógicos clássicos em poucos segundos, a partir de escolha aleatória de exemplos positivos e negativos. Também foi usado, com sucesso, para aprender regras que predizem estruturas secundárias de proteínas a partir da seqüência de aminoácidos, estrutura de relações nos projetos de drogas (remédios) e aprendizado de regras de diagnóstico temporal para sistemas físicos[BER96].

Segundo Bergadano e Gunetti[BER96], GOLEM possui algumas limitações:

- Encontrar um módulo próprio para o programa lógico; normalmente isto é feito fornecendo todos os exemplos com maior profundidade *h*, e esses exemplos nem sempre estão disponíveis;
- Não consegue aprender múltiplos predicados e
- Devido ao modelo utilizado para o programa lógico, GOLEM geralmente não é completo. Particularmente, a validade não é garantida pois se duas cláusulas são aprendidas independentemente e não cobrem exemplos negativos, a conjunção dessas cláusulas pode cobrir alguns desses exemplos.

## 4.5 FOIL

FOIL[QUI90,DZE93] estende algumas idéias de algoritmos de aprendizados do tipo atributo-valor para o paradigma ILP. Utilizada a abordagem de cobertura similar àquela utilizada pelo algoritmo AQ, que produz a descrição de um conceito no formato de regras de produção, no formato *Se.. Então*. Assim *Se* contém a descrição de um objeto, *Então* conterá a classe onde o objeto se encontra[MIC83]. FOIL realiza uma busca heurística baseada em informação similar

àquela utilizada em ID3, que também é um algoritmo que induz conceitos, porém no formato de árvore de decisão, neste caso, os nós representam os atributos e as folhas, os conceitos[QUI86].

A linguagem de hipóteses em FOIL é a linguagem de cláusulas livres de funções, i.e., significa que nenhum termo que não seja uma constante ou variável deve aparecer nas cláusulas induzidas. Fatos instanciados livres de função são usados para representar exemplos de treinamento e conhecimento prévio do domínio.

Depois de pré-processar um conjunto de treinamento, que consiste na geração de exemplos negativos, caso nenhum seja informado, o laço mais externo de FOIL repete os seguintes dois passos até todos os fatos serem cobertos:

- Encontra a cláusula que cobre alguns exemplos positivos e nenhum negativo e
- Remove os fatos cobertos por esta cláusula do conjunto de treinamento

Encontrar uma cláusula consiste de um certo número de passos de refinamento. A busca inicia com a cláusula $p(X_1 ,.... X_n ) \leftarrow$, a qual possui o corpo vazio. Em cada passo, a cláusula *c* construída é refinada adicionando uma literal em seu corpo. Essas literais são átomos positivos ou negativos na forma $X_i=X_j$ ou $q_k(Y_1,Y_2,...,Yn_k)$, onde os *X's* aparecem em *c*, os *Y's* aparecem em *c* ou podem ser novas variáveis, e $q_k$ é um predicado retirado do conhecimento prévio do domínio ou do predicado a ser aprendido *p*.

O laço de FOIL como apresentado produz uma definição de predicado (programa lógico) que é completo (cobre todos os exemplos positivos) e consistente (não cobre nenhum exemplo negativo). Quando trabalhando com dados imperfeitos, a completude e consistência necessitam ser relaxadas. Para parar a busca de literais a serem adicionadas a uma cláusula, FOIL emprega a codificação de restrição de comprimento que limita o número de *bits* necessários para explicitamente indicar o número de exemplos positivos cobertos pela cláusula. A construção de uma cláusula pára quando são cobertos apenas exemplos positivos ou quando não existem mais *bits* disponíveis para adicionar literais ao corpo da cláusula. A busca pelas cláusulas pára quando nenhuma nova cláusula pode ser construída sobre a restrição do tamanho disponível para a codificação, ou quando todos os exemplos positivos foram cobertos. A restrição da codificação existente degrada o desempenho de FOIL em relação à presença de dados "com ruído".

### 4.6 LINUS

LINUS[LAV94] é um sistema de aprendizagem ILP que incorpora a abordagem de sistemas de aprendizagem de atributo-valor. A idéia é transformar um classe restrita de problemas ILP na forma proposicional e resolver o problema transformado com um algoritmo de aprendizagem de atributo-valor. O valor do aprendizado proposicional é novamente transformado em linguagem de primeira ordem. Por um lado este método melhora o algoritmo de aprendizagem proposicional com o conhecimento prévio do domínio e uma linguagem de primeira ordem mais expressiva. Por outro lado, habilita as aplicações que tiveram sucesso no aprendizado proposicional na abordagem de primeira ordem. Como vários algoritmos de aprendizagem proposicional podem ser integrados e acessados via LINUS, LINUS é qualificado, também, como uma ferramenta ILP que oferece vários algoritmos de aprendizagem com suas qualidades específicas.

LINUS pode ser executado em dois modos: no modo de CLASS, correspondendo a um sistema de aprendizado proposicional ou no modo RELATION, onde se comporta como um sistema ILP.

O princípio básico de transformação de lógica de primeira ordem na forma proposicional é que todas as literais do corpo que são passíveis de aparecerem na cláusula de hipótese são determinadas, levando em consideração os tipos de variáveis. Cada uma dessas literais do corpo correspondem a um valor lógico no formalismo proposicional. Para um dado exemplo, o valor de seu argumento é substituído pelas variáveis de uma literal do corpo. Como todas as variáveis no corpo têm que aparecer na cabeça, a substituição indica um fato instanciado. Se for um fato verdadeiro, o exemplo correspondente da lógica proposicional será verdadeiro, caso contrário, falso. O resultado gerado pelo algoritmo de aprendizado proposicional é transformado para o formato de primeira ordem. A hipótese induzida é comprimida no passo de pós-processamento.

Para que a transformação na lógica proposicional e vice-versa sejam possíveis, algumas restrições na linguagem de hipóteses e conhecimento prévio do domínio são necessárias. Os exemplos de treinamento são fatos instanciados, podendo conter estruturas, mas não termos recursivos. Os exemplos negativos podem ser informados ou gerados pelo sistema LINUS através da hipótese do mundo fechado. LINUS oferece várias opções para controlar a geração de exemplos negativos.

A linguagem de hipótese do sistema LINUS é restrita para as cláusulas de banco de dados hierárquicos dedutivos, ou seja, programas tipados com predicados e tipos não recursivos, onde as variáveis do corpo são subconjuntos das variáveis da cabeça. Além de que funções, predicados e

cláusulas de hipóteses consistem de literais unificando duas variáveis (X=Y) e de literais com constantes associadas às variáveis (X=a). Certos tipos de literais podem aparecer na forma negada no corpo de uma hipótese.

O conhecimento prévio do domínio tem a forma de cláusula de banco de dados dedutível, ou seja, possíveis programas recursivos com variáveis tipadas. As definições dos tipos das variáveis são exigidas para serem não recursivas e devem ser informadas pelo usuário. O conhecimento prévio do domínio consiste de dois tipos de definição de predicado, as funções e os predicados nomeados. Funções nomeadas são predicados que computam um único valor de saída para um dado valor de entrada. O usuário tem que declarar os modos de entrada/saída. Quando ocorrer uma cláusula induzida, os argumentos de saída são instanciados para uma constante. Predicados nomeados são funções lógicas com apenas argumentos de entrada. Para uma dada entrada, esses predicados computam verdadeiro ou falso.

## 4.7 Considerações Finais

A Tabela 4.1 apresenta um resumo das características dos sistemas ILP estudados. Outros sistemas, e.g., CLINT, MOBAL, MERLIN, MIS, CLAUDIEN[RAE92, MOR93, BOS98, SHA83, RAE93], foram pesquisados também, porém, apenas os apresentados foram estudados mais profundamente para a implementação desse trabalho.

Tabela 4.1 – Resumo Características dos Sistemas Estudados

| **Sistema** | **Forma de Indução** | **Algoritmo de Busca** | **SQL** |
|---|---|---|---|
| Progol | *Inverse Entailment* | A* | não |
| RDT/DB | Top-Down (busca em grafo de refinamento) | sort lattice | sim |
| FILP | Top-Dowm (busca em grafo de refinamento) | não encontrado | não |
| Golem | Generalização Mínima Relativa | busca heurística | não |
| FOIL | Top-Down (busca em grafo de refinamento) | busca heurística | não |
| LINUS | Transformação do Problema para Lógica Proposicional | não encontrado | não |

Após o estudo realizado, Progol foi escolhido para servir como base para o sistema proposto por possuir uma interface relativamente amigável e uma boa documentação. Primeiramente foram realizados alguns experimentos com Progol para analisar suas características e forma de aprendizado. Em [LOP00] é apresentado o resultado desse estudo.

Após os experimentos realizados com o sistema Progol, foi evidenciado que este sistema não é apropriado para trabalhar com grande volume de dados já que todas as informações são

armazenadas na memória RAM. Apesar do armazenamento ser de forma codificada, ocupando menos espaço, esta abordagem limita o número de exemplos que o sistema poderia ter para a construção das hipóteses. Inicialmente, para contornar esta situação, foi proposto alterar o núcleo do sistema para acessar os exemplos a partir de um banco de dados relacional e executar a busca. Esta abordagem foi descartada pois além do *Progol* utilizar um mecanisno de *hashing* para codificar as cláusulas (dificultando o mapeamento para um comando SQL) seria necessário fazer uma manutenção nas funcionalidades de criação do grafo de busca e pesquisa para incluir as chamadas dos comandos *select*.

Devido às dificuldades encontradas para desenvolver uma versão do Progol que acessasse banco de dados relacional, foi necessário propor um sistema para desenvolver essa atividade. Assim foi desenvolvido o DBILP (***D**ata**B**ase miner based on **ILP***), que alia o poder de expressão de ILP com o poder de manipulação e gerenciamento de dados que um SGBD oferece através da linguagem SQL. Esse sistema é discutido em detalhes nos próximos capítulos.

# Capítulo 5

## 5 Implementação

Este capítulo apresenta como foi implementado o DBILP (***D**ata**B**ase miner based on **ILP***), um sistema baseado nos fundamentos de ILP que utiliza dados armazenados em um banco relacional para encontrar regras que descrevam estes dados. Primeiramente é apresentado o objetivo da construção desse sistema e em seguida, os módulos, a linguagem de entrada, o funcionamento e um exemplo de execução.

### 5.1 Objetivos

Conforme apresentado no capítulo anterior, após pesquisar alguns sistemas ILP, i.e., LINUS[LAV94], FILP[BER96], RDT/DB[BRO96], FOIL[QUI90], GOLEM[MUG90,MUS97] e Progol[MUG00], foi escolhido o último por, além de ser o mais recente, possuir um ambiente disponível para executar os programas ILP e boa documentação. Como Progol apresentou algumas deficiências que este trabalho propõe reduzir foi definido um novo sistema para melhorar as seguintes características ausentes em Progol e dos outros sistemas descritos no capítulo anterior :

- Trabalhar com grande volume de dados, e.g., *data warehouses*. Progol trabalha com os dados na memória principal do computador;
- Possibilitar o acesso a um banco de dados relacional. Progol trabalha com um arquivo texto;
- Possibilitar a utilização de múltiplas tabelas para construir as hipóteses; e
- Utilizar SQL como meio de acesso aos dados.

K. Shimazu e K. Furukawa em [SHI97] propuseram uma ferramenta que utiliza Progol e banco de dados relacional, porém essa ferramenta tem um módulo que gera um arquivo de entrada, baseado no banco de dados utilizado, que é reconhecido pelo sistema Progol. Portanto, tal característica ainda não resolve os problemas apontados anteriormente, pois o sistema Progol continua trabalhando no seu formato conforme foi concebido, utilizando arquivos textos com uma linguagem baseada na sintaxe reconhecida pelo Progol.

O sistema RDT/DB propõe recuperar dados armazenados em um banco utilizando SQL. DBILP e RDT/DB são bastante similares em suas propostas. Como não foi possível encontrar uma

implementação de RDT/DB de domínio público, não foi realizado experimentos para verificar o desempenho de ambos quanto à geração de regras, classificação e acesso ao banco de dados.

Assim, verificando as deficiências da maioria dos sistemas de aprendizado quanto à atividade de descoberta do conhecimento, principalmente Progol, foi proposto este novo sistema que, utilizando fundamentos de ILP, acessa diretamente os dados de uma banco de dados relacional utilizando SQL.

## 5.2 Módulos do Sistema DBILP

O DBILP possui, basicamente, três módulos principais, cujo relacionamento é apresentado na Figura 5.1.

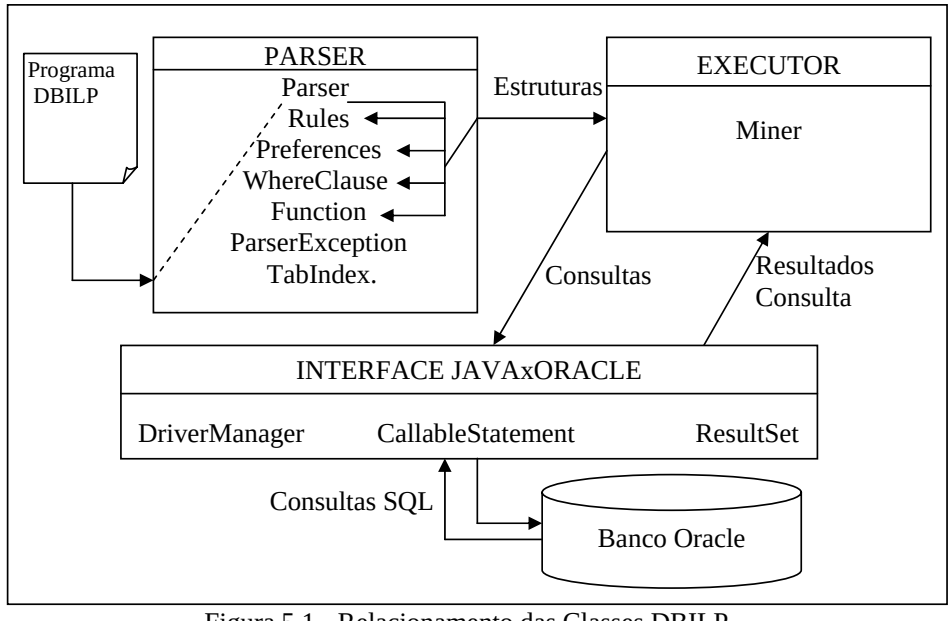



Figura 5.1 - Relacionamento das Classes DBILP

*Parser*: este módulo é responsável por reconhecer sintaticamente a linguagem definida para o sistema e criar as estruturas para transformar as definições DBILP em definições PL/SQL da Oracle®. É formado pelas classes:

> Parser: classe responsável para reconhecer sintaticamente o programa de entrada, gerando *tokens* para as outras classes;
>
> Rules: reconhece semanticamente a cláusula RULES do programa, gerando as estruturas para o Módulo Executor e os atributos que apareceram no comando *select* no momento de construção da *view*;

Preferences: reconhece semanticamente os comandos SET's da linguagem, gerando informações para o Módulo Executor;

WhereClause: reconhece semanticamente a cláusula RELATIONS de um programa DBILP. Gera a cláusula *where* do comando *select* que gera a *view*;

Function: reconhece semanticamente as função declaradas. Mapea essas funções no formato Oracle.

ParserException: classe que trata as exceções das classes deste módulo; e

TabIndex: cria as estruturas das tabelas declaradas, i.e., as condições do *join* entre as tabelas, quais tabelas são as tabelas pais, entre outros.

*Executor*: módulo responsável por achar a(s) hipótese(s) que melhor descreverá(ão) o conjunto de exemplos do banco de dados. Esta busca é feita baseada na linguagem de entrada, i.e., definição das tabelas, cabeça e corpo das hipóteses. Utiliza a classe de interface JAVAxORACLE para acesso ao banco de dados. Formado pela classe Miner.

*Interface JAVAxORACLE*: este módulo é responsável pelo acesso do sistema DBILP ao banco de dados relacional. Formado pelas classes *DriverManager*, *CallableStatement* e *ResultSet*.

Todos os módulos e suas respectivas classes foram desenvolvidos utilizando o JDK 1.1.8 no ambiente JDeveloper 3.0® da Oracle e os dados estão armazenados em um banco de dados relacional gerenciados pelo SGDB Oracle®.

O DBILP possui uma linguagem para a definição dos predicados que serão utilizados para encontrar a melhor hipótese. Esta linguagem será apresentada na próxima seção.

### 5.3 Linguagem de Definição

DBILP utiliza as tendências declarativa, para definir o espaço de hipóteses a ser considerado pelo aprendiz (o que procurar); e de preferência, que determina como procurar no espaço de hipótese e qual hipótese focar[MUG94]. Estas tendências são implementadas através da linguagem de entrada definida para o DBILP. Nesta linguagem são definidas quais tabelas estarão envolvidas na busca e qual é o conceito objetivo.

A sintaxe da linguagem de entrada de DBILP é apresentada a seguir:

[SET <CONFIDENCE num>|<SUPPORT num>|<VIEW ON|OFF>]
RELATIONS
<TabelaPai.atributo$_1$> : <TabelaFilho.Atributo$_1$>
: :
<TabelaPai.atributo$_n$> : <TabelaFilho.atributo$_n$>
RULES
H: <hipótese> (<[&]tabela.atributo>|<FCT(<nome func>([<par>]))>,...)
$B_1$: <nome_predicado$_1$> (<[&]tabela.atributo>|<FCT(<nome func>([<par>]))>,...)
:
$B_n$: <nome_predicado$_n$> (<[&]tabela.atributo>|<FCT(<nome func>([<par>]))>,...)
[FUNCTIONS
<nome_func$_1$> : <par>
<def função>
:
<nome_func$_n$> : <par>
<def função>]

Onde:

- Os comandos SET indicam configuração do programa. Os modificadores CONFIDENCE e SUPPORT não foram implementados pois optou-se pela busca exaustiva dos conceitos. Estes modificadores indicam a confiança e o suporte que o usuário definiria para a hipótese gerada [FAY98]. O modificador VIEW indica se a visão criada será materializada (ON) ou não (OFF). As configurações são opcionais, neste caso é assumido o valor padrão ON para VIEW;

- A seção RELATIONS indica quais tabelas serão utilizadas para a criação da visão e quais são as relações entre elas. Esta seção poderia não ser implementada já que os SGBD possuem dicionários de dados com essas informações porém, nesta linguagem, o usuário pode informar atributos que não são chaves para criar um *join* entre as tabelas envolvidas. No momento da criação da visão baseada nas relações informadas não existe a preocupação com a perda da informação[WID97];

- A seção RULES declara os átomos/predicados que farão parte da(s) hipótese(s). Todos os argumentos dos predicados deverão estar no formato: <*nome_tabela*>.<*nome_atributo*>, pois, como apresentado anteriormente, todos os argumentos deverão estar armazenados em uma tabela. Esta seção é similar a seção MODES de Progol, inclusive a forma de declarar quais termos serão instanciados e quais serão substituídos por variáveis.

- O prefixo H indica que o predicado informado será a cabeça da hipótese a ser encontrada. Este predicado deverá ser binário e possuir uma variável, além de poder ser declarado apenas um H em um determinado programa DBILP;
- O prefixo B indica os predicados candidatos ao corpo da(s) hipótese(s) candidata(s);
- O prefixo & antes de um argumento de um predicado indica que este valor será substituído por uma variável no momento da generalização da hipótese candidata;
- O prefixo FCT especifica que na posição indicada dentro do predicado existirá uma função que deverá ser declarada na seção FUNCTIONS. A função declarada poderá ter, no máximo, um parâmetro e retornará, no máximo, um valor;
- Na seção FUNCTION todas as funções declaradas em RULES deverão ser definidas. A definição de uma função deverá ser feita de duas maneiras: livre, onde utiliza-se comandos PL/SQL® do SGBD Oracle®, ou proprietária, onde utiliza-se a linguagem definida por DBILP para implementar as funções; e
- A linguagem proprietária possui os seguintes elementos sintáticos:
  - Operadores relacionais: =, <>, >, <, >=, <=, igual, diferente, maior, menor, maior ou igual, e menor ou igual, respectivamente.
  - O valor de retorno que deverá ser um valor alfanumérico (o valor deverá estar entre apóstrofes)
  - Assim um comando proprietário do DBILP será uma comparação entre o parâmetro com um valor qualquer, retornando um valor alfanumérico. Este comando será transformado em um comando PL/SQL no formado IF <condição> THEN RETURN <valor> END IF;

A linguagem apresentada permite que DBILP tenha um objetivo no momento da mineração dos dados, trabalhando apenas com os dados e tabelas definidos no programa a ser executado.

### 5.4 Funcionamento do DBILP

DBILP utiliza os mecanismo de referências para reduzir o espaço de busca das hipóteses. Este utilização é realizada graças a sua linguagem de entrada. Após a linguagem de entrada ser reconhecida pelo *Parser*, são construídos todas as estruturas que irão guiar o *Executor* na busca das hipóteses.

O mecanismo básico de aprendizado do DBILP, baseado na linguagem de hipóteses definida no programa de entrada, executa especializações e generalizações sucessivas até encontrar o conjunto de hipóteses que descrevam os dados. Inicia com *H:-Bn*, onde *H* é o conceito a ser aprendido e *Bn* é um dos candidatos ao corpo do conceito. Se necessário acrescenta outros literais declarados, até que nenhum exemplo negativo seja coberto.

A especialização é feita baseada na cabeça e nos corpos definidos para a hipótese candidata. O *Executor* constrói a hipótese baseado na escolha dos corpos candidatos, em seguida, consulta o banco de dados para instanciar todos os argumentos dos predicados da hipótese selecionada. Após construir uma hipótese especializada, o DBILP generaliza esta hipótese baseado nas informações de substituição dos argumentos por variáveis definidas no programa de entrada. Neste ponto, a hipótese é generalizada, então o *Executor* constrói um comando SQL para verificar o número de exemplos cobertos pela hipótese. O DBILP não trabalha com exemplos negativos, portanto, para validar a hipótese, constrói um comando SQL buscando todos os exemplos que descrevem a classe em questão, não colocando esta classe na cláusula *where*. Se o comando retornar algum exemplo cuja classe não pertence a classe em questão, indica que a hipótese cobriu falsos exemplos positivos, portanto é retirada do conjunto de hipóteses.

Todas as vezes que uma hipótese é verificada no banco de dados, existe uma transformação da mesma para o formato SQL. Para tal, o atributo que descreve a classe é colocado na primeira parte do comando *select*, os predicados corpos são colocados, quando aplicado, na cláusula *where*. A seguir é apresentado um exemplo desta transformação:

```
hipótese: classe_carro(A,mboa):-val_compra(A,medio), seguranca(A,alta)

Comando SQL:      select v.classe, count(*) into vclass, vtotal
                  from visao_criada v
                  where v.preco='medio'
                  and v.seguranca='alta'
                  group by v.classe;
```

Este comando popula as variáveis *vclass* e *vtotal* que possuem, respectivamente, o nome da classe e a quantidade de exemplos cobertos por *v.preco='medio' and v.seguranca='alta'*. Se o comando SQL gerado recuperar apenas uma linha e nesta linha *vclass* coincidir com o a classe a ser descrita, a regra é armazenada no conjunto de regras candidatas.

Devido ao fato de utilizar comandos SQL para acessar o banco, todas as consultas ordenam as tabelas por classe a ser descrita, protanto as regras geradas pelo DBILP são criadas de forma ordenada, assim a ordem dos exemplos não influencia no aprendizado.

Após a criação do conjunto de hipóteses candidatas, o sistema verifica quais hipóteses são subjugadas. Iniciando com as hipóteses com menor tamanho, ou seja, menos predicados no corpo, é verificado se esta hipótese subjuga alguma outra, sendo que as subjugadas são retiradas do conjunto. Após esse processo, o conjunto resultante é apresentado.

A seguir são apresentados todos os passos que DBILP realiza para descrever um conjunto de conceitos a partir de uma linguagem de entrada e um banco de dados:

1. Baseado em RELATIONS e RULES cria a visão. A seção RELATIONS da linguagem definida para DBILP é utilizada para fazer o *join* entre as tabelas envolvidas e a seção RULES define quais atributos serão utilizados na visão. Os atributos na visão receberão o nome *An* onde *n* é a posição do atributo dentro da definição em RULES. Como a cabeça da regra é a primeira a ser definida e é binária, os conceitos a serem descritos sempre serão *A1* e *A2* na visão.

2. Caso um atributo seja uma função, esta é criada antes da criação da visão, assim os valores daquele atributo serão substituídos pelo retorno da função declarada;

3. Calcula quantas combinações poderão ser feitas com o número de predicados candidatos declarados ao corpo. Utiliza a seguinte fórmula:

$$\frac{\textbf{\textit{fatorial}}(QtdeCandidatos)}{\textbf{\textit{fatorial}}(QtdeCandidatos\text{-}QtdedeCombinações) \times \textbf{\textit{fatorial}}(QtdeCombinações)}$$

   sendo que *QtdeCombinações* inicia com 1 indo até *QtdeCandidatos*. O resultado para cada *QtdeCombinações* é somado com o próximo, sendo que o resultado final será o número de combinações. Para 4 candidatos ao corpo da regra teríamos a seguinte combinação:

$$\frac{4!}{(4-1)!\times 1!} + \frac{4!}{(4-2)!\times 2!} + \frac{4!}{(4-3)!\times 3!} + \frac{4!}{(4-4)!\times 4!}$$

   Assim 4+6+4+1 é igual a 15. Portanto 15 seria o número de combinações a serem realizados com 4 candidatos ao corpo. Essas combinações seriam arranjados da seguinte forma:

   - $B_1$;
   - $B_2$;

- $B_3$;
- $B_4$;
- $B_1,B_2$;
- $B_1,B_3$;
- $B_1,B_4$;
- $B_2,B_3$;
- $B_2,B_4$;
- $B_3,B_4$;
- $B_1,B_2,B_3$;
- $B_1,B_2,B_4$;
- $B_1,B_3,B_4$;
- $B_2,B_3,B_4$; e
- $B_1,B_2,B_3,B_4$.

4. Agrupa os exemplos pelas classes descritas em H;
5. Seleciona a próxima linha exemplo da visão;
6. Cria a cláusula mais específica com a linha selecionada baseada nas combinações geradas, i.e., H:-$B_1$, em seguida H:-$B_2$, e assim por diante, até H:- $B_1,B_2,B_3,B_4$. As combinações param caso alguma combinação cubra apenas exemplos positivos, não gerando falsos negativos ou falsos positivos; e
7. Executa os passos 5 e 6 até não existirem mais exemplos candidatos (não cobertos) na visão criada para aquela classe ou quando uma hipótese cobrir todos os exemplos da classe, excluindo aquelas linhas já cobertas por alguma hipótese candidata.

Como pode ser percebido, a busca pela melhor hipótese é feita de forma exaustiva, ou seja, todos exemplos positivos são utilizados para gerar hipóteses candidatas. O Apêndice A - Algoritmos apresenta os algoritmos principais, em pseudo código, para encontrar as hipóteses candidatas: aquele que encontra as hipóteses candidatas e aquele que retira as hipóteses subjugadas do conjunto gerado pelo primeiro algoritmo.

A linguagem de entrada de DBILP força algumas restrições para que a implementação fosse possível. Essas restrições são as seguintes:

1. A cabeça da hipótese deverá ser binária, sendo que um dos termos deverá ser substituível por uma variável;
2. A cabeça e os corpos candidatos as hipóteses deverão ser seguras e bem formadas [ABI95]; e
3. Não aceita recursividade, ou seja, o predicado que aparece como cabeça não pode aparecer como corpo.

Além das restrições impostas pela sintaxe da linguagem de entrada, o *Executor* impõe as seguintes restrições:

1. Trabalha com a hipótese do mundo fechado (CWA – Closed World Assumption) [ABI95],ou seja, não existem exemplos negativos ou todos os exemplos que não são positivos são considerados negativos; e
2. Os exemplos utilizados deverão estar armazenados em tabelas.

Para clarificar um pouco mais o funcionamento do sistema DBILP, a próxima seção apresenta um exemplo de execução do sistema com dados reais.

## 5.5 Exemplo de Execução

Esta seção apresenta um exemplo do funcionamento do sistema DBILP. Este exemplo é um subconjunto do conjunto ou base de dados *car* obtida do repositório de aprendizado de máquina da UCI [BLA98].

### 5.5.1 Formato e Transformação dos Dados

O conjunto utilizado apresenta valores que indicam as possíveis classes de um carro. As classes a serem generalizadas são: não aceitável (unacc), aceitável (acc), boa (good) e muito boa (v-good). Para caracterizar estas classes os seguintes atributos são considerados:

- Preço de venda – *buying (‘v-high’,’high’,’med’,’low’)*;
- Custo de manutenção – *maint (‘v-high’,’high’,’med’,’low’)*;
- Número de portas – *doors (‘2’, ‘3’, ‘4’, ‘5-more’)*;
- Capacidade de pessoas – *persons (‘2’, ‘4’, ‘more’)*;

- Capacidade do porta-malas – *lug_boot (‘big’, ‘med’, ‘small’)*; e
- Grau de segurança – *safety (‘high’, ’med’, ‘small’)*.

O conjunto *car* está no formato texto, onde cada linha representa a informação sobre um carro. O conteúdo desse arquivo texto foi modelado para que pudesse ser armazenado em tabelas, gerando o modelo entidade relacionamento (MER) (notação Buchmann[BAR94]) apresentado na Figura 5.2.

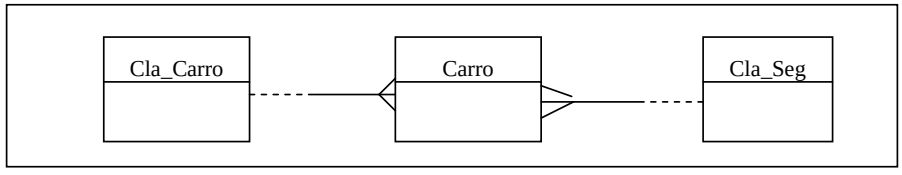



Figura 5.2 – Modelos Entidade Relacionamento para o Conjunto *car*

As tabelas presentes na Figura 5.2 armazenam os seguintes dados:

Cla_Carro: o domínio das classes dos carros. Atributos: cod_classe e classe.

Carro: armazena cada linha do arquivo texto com os dados de um carro. Os carros foram nomeados, seqüencialmente, *carnnn*, onde *nnn* é o número da linha das informações sobre um carro. Esta tabela armazena todas as informações sobre a base *car*, exceto sua classe e grau de segurança. Atributos: CodCar, NomCar, ValVen, ValMan, NumDoo,NumPer, CapLug, CodSaf e CodCla.

Cla_Seg: armazena o domínio de valores para o grau de segurança de um carro. Atributos: Cod_Seg e Descr.

Os únicos atributos a serem normalizados desse conjunto de dados foram aqueles que representavam a classe e o nível de segurança do carro. Esta escolha foi feita baseado na forma que projetista de sistemas experientes solucionariam este problema. Normalmente os atributos numéricos que representam valores reais não são normalizados, mesmos que seus valores se repitam e g, preço do carro, custo de manuntenção, capacidade do bageiro.

O preço de compra, o valor de manutenção e a capacidade do bagageiro foram transformados em valores numéricos e definido qual a faixa de valores se encaixava na classificação. O progama apresentado na próxima sub-seção apresenta como a classificação foi discretizada (seção Functions do programa exemplo).

### 5.5.2 Programa DBILP

O programa escrito para trabalhar com o conjunto acima é o seguinte:

```
Set View On
Relations
   cla_carro.cod_classe : carro.cod_classe
   cla_seg.cod_seg      : carro.cod_seg
Rules
   H: class(&carro.nome,cla_carro.classe)
   B: buying(&carro.nome,FCT(CVenda(carro.val_venda)))
   B: maint(&carro.nome,FCT(CManut(carro.val_manut)))
   B: doors(&carro.nome,carro.n_porta)
   B: persons(&carro.nome,carro.capac_pessoas)
   B: lug_boot(&carro.nome,FCT(CBoot(carro.capac_bagag)))
   B: safety(&carro.nome,cla_seg.descr)
Functions
   CBoot: a
         a > 300  'big'
         a > 150  'med'
         a <= 150 'small'
   CVenda: a
         a > 100000 'v-high'
         a > 50000  'high'
         a > 20000  'med'
         a <= 20000 'low'
   CManut: a
         a > 10000 'v-high'
         a > 5000  'high'
         a > 2000  'med'
         a <= 2000 'low'
```

Após o módulo *Parser* reconhecer este programa, o módulo *Executor* gera os seguintes programas PL/SQL.

Criação das funções definidas:

```
Create or Replace Function FCBoot (A Number) Return VarChar2 Is
Begin
      If a > 300 then return 'big'; end if;
      If a > 150 then return 'med'; end if;
      If a <=150 then return 'small'; end if;
End;
/
Create or Replace Function FCVenda (A Number) Return VarChar2 Is
Begin
      If a > 100000 then return 'v-high'; end if;
      If a > 50000  then return 'high'; end if;
      If a > 20000  then return 'med'; end if;
      If a <= 20000 then return 'low'; end if;
End;
/
Create or Replace Function FCManut (A Number) Return VarChar2 Is
Begin
      If a > 10000 then return 'v-high'; end if;
      If a > 5000  then return 'high'; end if;
      If a > 2000  then return 'med'; end if;
      If a <= 2000 then return 'low'; end if;
End;
/
```

Criação da visão:

```
Create or Replace View Tclass As
Select c.nome a1, c1.classe a2,
       FCVenda(c.val_venda) a3, FCMaint(c.val_manut) a4,
       c.n_porta a5, c.capac_pessoas a6,
       FCBoot(c.capac_bagag) a7, c2.descr a8
From   Carro c, cla_carro c1, cla_seg c2
Where  c1.cod_classe=c.cod_classe
And    c2.cod_seg = c.cod_seg
Order  by 1,2;
```

A Tabela 5.1 e a Tabela 5.2 apresentam o subconjunto retirado da base *car* utilizado como fonte para a geração da visão TCLASS, conforme comando *select* gerado pelo DBILP (Tabela 5.3).

Tabela 5.1 - Conteúdo da tabela CLA_CARRO e CLA_SEG

| **CLA_CARRO** | |
|---|---|
| Cod_Classe | Classe |
| 1 | unacc |
| 2 | acc |
| 3 | good |
| 4 | v-good |

| **CLA_SEG** | |
|---|---|
| Cod_Seg | Descr |
| 1 | high |
| 2 | med |
| 3 | low |

Tabela 5.2 – Conteúdo da tabela CARRO

| **CARRO** | | | | | | | |
|---|---|---|---|---|---|---|---|
| Nome | Cod_Classe | Val_Venda | Val_Manut | N_Portas | Capac_Pessoa | Capac_Bagag | Cod_Seg |
| car1 | 1 | 60000 | 3100 | 2 | 2 | 150 | 2 |
| car2 | 1 | 65000 | 3200 | 2 | 2 | 150 | 1 |
| car3 | 1 | 53000 | 3650 | 2 | 2 | 220 | 3 |
| car4 | 1 | 57000 | 4200 | 2 | 2 | 255 | 2 |
| car5 | 1 | 59500 | 3110 | 2 | 2 | 320 | 3 |
| car6 | 1 | 75000 | 3680 | 2 | 2 | 330 | 1 |
| car202 | 2 | 51000 | 1200 | 3 | 4 | 350 | 2 |
| car203 | 2 | 55500 | 1250 | 3 | 4 | 320 | 1 |

Tabela 5.3 – Visão gerada pelo DBILP

| **TCLASS (Visão)** | | | | | | | |
|---|---|---|---|---|---|---|---|
| A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 |
| car202 | acc | high | low | 3 | 4 | big | med |
| car203 | acc | high | low | 3 | 4 | big | high |
| car1 | unacc | high | med | 2 | 2 | smal | med |
| car2 | unacc | high | med | 2 | 2 | smal | high |
| car3 | unacc | high | med | 2 | 2 | med | low |
| car4 | unacc | high | med | 2 | 2 | med | med |
| car5 | unacc | high | med | 2 | 2 | big | low |
| car6 | unacc | high | med | 2 | 2 | big | high |

### 5.5.3 Execução do DBILP

A seguir é apresentado os passos de execução do programa DBILP dado (veja Subseção 5.5.2). Essas atividades são executadas pelos módulos *Executor* e *Interface JAVAxORACLE*:

1. Conta e armazena quanto exemplos possui cada classe, neste exemplo, *acc* possui 2 e *unacc* possui 6.
2. Escolhe uma linha para iniciar a busca da hipótese candidata (primeira linha)
3. Verifica se a linha selecionada já foi coberta por alguma hipótese gerada. Caso tenha sido, vota para o passo 2, senão continua.
4. Escolhe a próxima combinação das combinações possíveis, i.e., H:-$B_1$.
5. Gera a cláusula mais específica *class(car202,acc):-buying(car202,high)*. Esta cláusula serve para guiar a busca da hipótese candidata. Isto é feito substituindo os termos por variáveis, obtendo, assim, *class(A,acc):-buying(A,high)*.
6. Constrói o seguinte comando *select*: *select A2, count(*) from TCLASS where A3='high'*. Este comando cobre todos os dois exemplos da classe *acc*, porém, cobre 6 exemplos para *unacc*. Assim esta hipótese é descartada.
7. Busca a próxima combinação, i.e., H:-$B_2$, assim gera, *class(car202,acc):-maint(car202,low)*.
8. Utiliza esta cláusula para gerar *class(A,acc):-maint(A,low)*, gerando o comando *select*, *select A2,count(*) from TCLASS where A4='low'*. Este comando cobre todos os dois exemplos da classe *acc* 2 e nenhum para *unacc*. Assim a hipótese *class(A,acc):-maint(A,low)*, é escolhida para descrever a classe *acc*.
9. Executa os mesmos passos para a próxima classe, i.e., *unacc*.
10. executa passos similares aos da classe *acc*, chegando na hipótese *class(A,unacc):-maint(A,med)*, que gera o comando *select*, *select A2,count(*) from TCLASS where A4='med'*. Este comando cobre os 6 exemplos para *unacc*, i.e., 100% dos casos, e 0 para *acc*.

11. Apresenta as hipóteses ***class(A,acc):-maint(A,low)*** para descrever carros do tipos aceitáveis (*acc)* e ***class(A,unacc):-maint(A,med)*** para descrever carros do tipo inaceitáveis (*unacc*).

O exemplo apresentado tem dados que facilitam a busca da(s) hipótese(s) candidatas, porém, em uma situação real, DBILP escolheria todas as cláusulas que cobrem apenas exemplos positivos e, após finalizar a busca, procura as cláusulas que são subjugadas pelas cláusulas mais genéricas, retirando as primeiras da lista de hipóteses candidatas.

## 5.6 Considerações Finais

Neste capítulo foi apresentado o sistema proposto, DBILP. Este sistema foi concebido para aplicar a técnica de ILP em banco de dados relacionais envolvendo múltiplas tabelas e também grande volume de dados. O sistema Progol foi base para a definição do DBILP, i.e., a sua linguagem de programação, a parte da forma de encontrar a(s) hipótese(s) candidata(s), ou seja, especializando a hipótese e em seguida generalizado para torná-la candidata. Optou-se por não utilizar a técnica de *Inverse Entailment* na primeira versão do sistema pois não havia tempo hábil par implementá-la. Assim foi empregado a busca de  forma exaustiva, assim, todos os exemplos e predicados definidos são considerados para construir o conjunto de hipóteses.

DBILP se destaca dos sistemas descritos no Capítulo 4 pelo fato de trabalhar com os dados diretamente no banco de dados relacional, delegando para o SGBD Oracle o trabalho de manipular os dados e gerenciamento da memória principal do computador. Utiliza comandos SQL para iniciar o processo de generalização e verificação de cobertura de exemplos positivos da hipótese candidata.

Foi apresentado um exemplo de execução do DBILP. Os dados utilizados foram um subconjunto da base *car*. Neste exemplo foi mostrado todos os passos envolvidos para a busca da hipótese, desde a criação do programa, até a execução de todos os módulos do DBILP.

O próximo capítulo apresenta um experimento entre o sistema proposto e os sistemas Progol, C4.5 e CN2, o primeiro foi descrito na Seção 4.1 e os dois últimos foram apresentados brevemente na Seção 2.2.1 pois são sistemas bastantes conhecidos no meio científico. Esta comparação tem a finalidade de validar as hipóteses geradas pelo sistema DBILP e verificar o comportamento do sistema implementado com relação a um sistema também baseado em ILP e dois outros orientados a atributo-valor.

# Capítulo 6

## 6 Resultados do Experimento

Este capítulo apresenta um experimento realizado com o sistema DBILP com o objetivo de verificar o comportamento dele em relação a outros três sistemas bem conhecidos para induzir regras (conceitos). Os sistemas escolhidos foram CN2 e C4.5, orientados a atributo-valor, e Progol, um sistema ILP. Este experimento foi baseado em um trabalho anterior [LOP00], o qual utilizou os três sistemas citados anteriormente. Foram incluídos apenas os resultados do sistemas DBILP.

### 6.1 Descrição dos Conjuntos de Dados

Para realizar o experimento entre estes sistemas foram utilizados três conjuntos de dados: *car*, *nursery* e *zoo*, extraídas a partir do UCI Machine Learning Repository[BLA98], que apresentam um diversificação considerável de número de classes, tamanhos, e número e tipo de atributos, sumarizados na Tabela 6.1.

Tabela 6.1 - Descrição do conjunto de dados

| Bases | Tamanho | Classes | | | | | | | | | Atributos | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Número | Distribuição de Exemplos | | | | | | | | Numéricos | Booleanos | Discretos |
| zoo | 101 | 7 | 41 | 20 | 5 | 13 | 4 | 8 | 10 | | 2 | 15 | 1 |
| car | 1728 | 4 | 384 | | 69 | | 65 | | 1210 | | | | 6 |
| Nursery | 12960 | 5 | 4320 | | 2 | 328 | | 4266 | | 4044 | | | 8 |

Esses dados estão no formato proposicional e não contém atributos ausentes. Para aplicar os Progol e DBILP, algumas adaptações tiveram que ser realizadas. Para Progol os dados tiveram que ser incluídos em um programa Progol, separando-os em conhecimento prévio do domínio e exemplos positivos. Para DBILP, tiveram que ser convertidos para o formato relacional e armazenados em tabelas. A Figura 5.2 do Capítulo 5 apresenta o MER para o conjunto *car* e a Figura 6.1 *a* e *b* apresentam, respectivamente, os MER para os conjuntos *zoo* e *nursery*. Como este experimento é baseado no experimento realizado por Lopes[LOP00] não foi possível escolher um conjunto de dados grande para demonstrar a capacidade do DBILP de trabalhar com grandes volumes de dados.

O conjunto *zoo* possui dados que caracterizam animais, tais como, número de pernas, se mamam, se possuem penas, entre outros, classificando os animais como mamíferos, pássaros, peixes, répteis, anfíbios, crustáceos e insetos. O conjunto *nursery* possui dados que caracterizam pedidos para ingresso a um berçário, tais como, situação financeira dos pais, se têm problemas

sociais, número de filhos, entre outros, classificando esses pedidos como: não recomendado, recomendado, prioritário, bastante recomendado e especialmente recomendado.

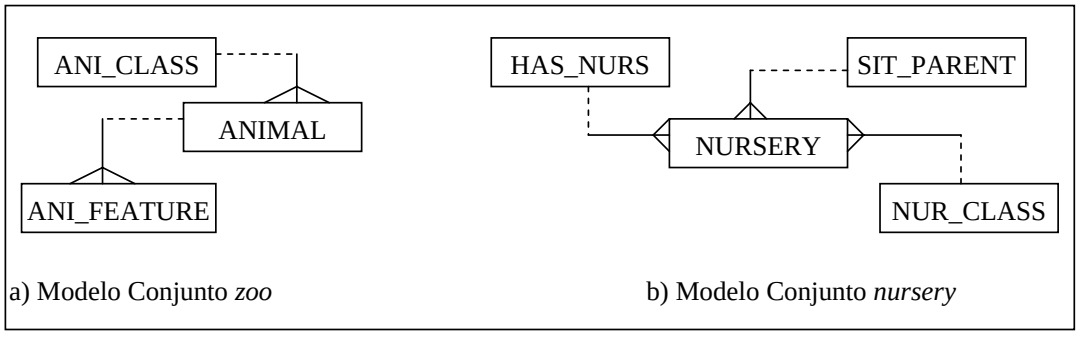



Figura 6.1 – Modelo Entidade Relacionamento para os Conjuntos *zoo* e *nursery*

A definição de quais atributos se tornariam tabelas e quais não, foi realizada baseada na experiência de profissionais que trabalham em projetos de sistemas que utilizam banco de dados relacionais.

## 6.2 Descrição do Experimento

O experimento realizado neste trabalho, conforme descrito anteriormente, seguiu os mesmos passos descritos em[LOP00]. A seguir é apresentada a metodologia utilizada.

O experimento foi dividido em duas partes:

1. Geração das curvas de aprendizado, utilizando-se apenas conjunto de dados *car* e
2. Avaliação dos desempenhos dos sistemas utilizando-se todos os três conjuntos.

A definição de quais conjuntos de dados seriam escolhidos para o experimento foi baseada na escolha realizada por Lopes[LOP00] . As etapas para a geração das curvas de aprendizado foram as seguintes:

1. Obtenção aleatória do conjunto de treinamento, 80% do casos, e conjunto de teste (20% dos casos), mantendo a distribuição da população original;
2. A partir do conjunto de treinamento, foram gerados aleatoriamente treze novos conjuntos de treinamento, estes contendo, respectivamente, 5%, 10%, 15%, 20%, 25%, 30%, 40%, 50%, 60%, 70%, 80%, 90% e 100% dos exemplos do conjunto de treinamento; e

3. Obtenção da curvas de aprendizado através da aplicação dos quatro sistemas (CN2, C4.5, Progol e DBILP) nos treze conjuntos de treinamento, confrontando as regras geradas para cada conjunto de treinamento com o conjunto de teste. As curvas geradas são apresentadas na Figura 6.2.

As etapas para avaliação do desempenho dos sistemas foram as seguintes:

1. Mantendo a distribuição estatística das classes, foram gerados aleatoriamente o conjunto de teste utilizando-se 20% dos exemplos de cada conjunto. Os exemplos remanescentes (85%) foram utilizados para o treinamento. Foram gerados 10 pares distintos desses conjuntos para cada base de dados.
2. Aplicação dos resultados de cada sistema no conjunto de teste, obtendo-se a média de desempenho de classificação de cada sistema e suas respectivas médias dos números de regras geradas.

A Tabela 6.2 apresenta o desempenho de classificação de cada sistema e está organizada da seguinte forma:

- A coluna *Bases* contém os três conjuntos utilizados;
- A coluna *Classes*, as classes que deverão ser induzidas para cada conjunto;
- As colunas *Progol*, *CN2*, *C4.5* e *DBILP* possuem o número de exemplos cobertos de forma correta e incorreta pelas regras geradas para cada classe, de cada sistema, respectivamente;
- A linha *Total*, indica o total de acerto e erro para cada sistema;
- A linha *Não* indica se houve algum exemplo não coberto; e
- A linha *Acerto* apresenta, em percentual, a média de acerto sobre todos os experimentos, além do desvio padrão.

A Figura 6.3 apresenta, graficamente, os resultados apresentados nesta tabela. Esta figura está organizada da seguinte forma:

- O eixo *x* representa o percentual de acertos das regras geradas pelo conjunto de treinamento em relação ao conjunto de teste. Este eixo inicia em 50% para que a escala fique mais precisa;

- O eixo *y* representa o tamanho, em percentual, do conjunto de treinamento utilizada para gerar as regras que foram confrontadas com o conjunto de testes; e
- As linhas dentro do gráfico reprentam o comportamento de cada sistema no aprendizado.

### 6.3 Análise dos Resultados

A Figura 6.2 apresenta as curvas de aprendizado geradas a partir da primeira parte do experimento. Pode-se observar que os sistemas DBILP e CN2 tiveram um desempenho superior aos sistemas Progol e C4.5. Também é observado que, conforme o conjunto de aprendizado aumenta em relação ao número de exemplos, o desempenho de todos os sistemas cresce também, conforme esperado. Mesmo apesar do sistema Progol não ter uma curva tão suave quanto os outros sistemas.

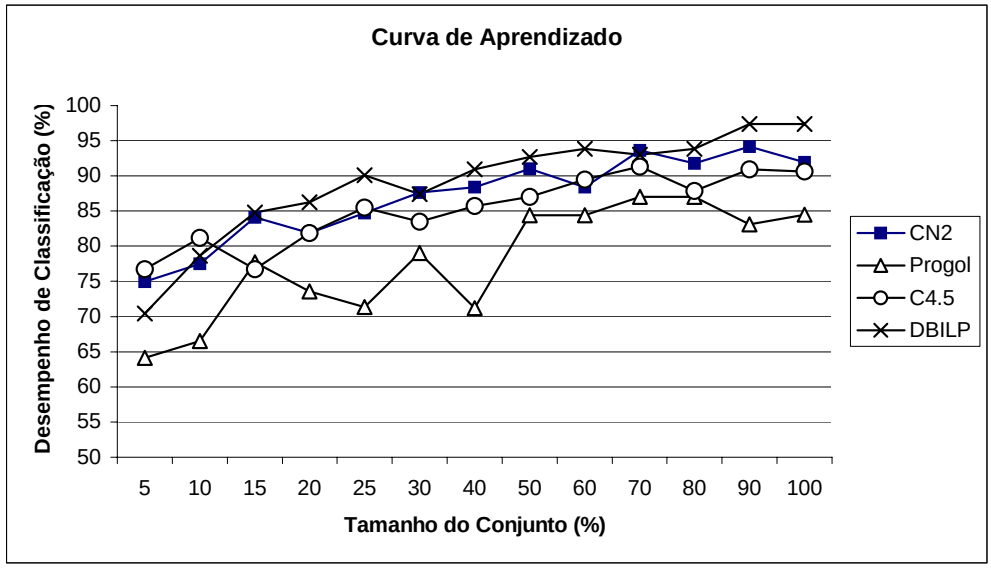



Figura 6.2 - Curva de Aprendizado do Experimento

Baseado ainda na Figura 6.2, pode-se observar que depois que o conjunto de treinamento foi superior à 15%, DBILP teve melhor desempenho que os outros sistemas, chegando a quase 100% de desempenho para o último conjunto de treinamento.

A segunda parte do experimento realizado com os quatro sistemas foi a verificação dos desempenhos dos sistemas em relação à classificação e número de regras geradas para os três conjuntos, *car*, *zoo* e *nursery*.

Através da Tabela 6.2 pode-se verificar que os desempenhos de classificação do C4.5, CN2 e DBILP são bastante semelhantes. Porém o sistema Progol obteve os piores resultados, ao redor 90% e com um considerável desvio padrão.

Ainda observando a Tabela 6.2, verifica-se que a classe *priority* da base *nursery* foi a que mais apresentou dificuldades para ser classificada por todos os sistemas, gerando muitos falsos exemplos positivos (coluna *Incor*.).

Tabela 6.2 - Avaliação dos sistemas para o conjunto de teste

| Bases | Classes | Progol | | CN2 | | C4.5 | | DBILP | |
|---|---|---|---|---|---|---|---|---|---|
| | | Corretas | Incor. | Corretas | Incor | Corret | Incor. | Corretas | Incor. |
| *zoo* | *1* | 7 | 1 | 8 | 0 | 8 | 0 | 8 | 0 |
| | *2* | 4 | 0 | 4 | 0 | 4 | 0 | 4 | 0 |
| | *3* | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| | *4* | 2 | 1 | 3 | 0 | 3 | 0 | 3 | 0 |
| | 5 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| | 6 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 0 |
| | 7 | 1 | 1 | 2 | 0 | 2 | 0 | 2 | 0 |
| | Total | 15 | 6 | 18 | 3 | 19 | 2 | 21 | 0 |
| | Não | 0 | | 0 | | 0 | | 0 | |
| | Acertos | 71,43 % + 9,86 | | 85,71% + 0,00 | | 90,48% + 0,00 | | 98,10% + 3,81 | |
| *car* | *acc* | 60 | 17 | 71 | 6 | 64 | 13 | 74 | 2 |
| | *good* | 6 | 8 | 13 | 1 | 10 | 4 | 12 | 1 |
| | *unacc* | 231 | 11 | 237 | 5 | 238 | 4 | 241 | 1 |
| | *vgood* | 5 | 8 | 12 | 1 | 7 | 6 | 11 | 0 |
| | Total | 302 | 44 | 333 | 13 | 319 | 27 | 338 | 4 |
| | Não | 0 | | 0 | | 0 | | 0 | |
| | Acertos | 87,28% + 2,25 | | 96,24% + 0,00 | | 92,2% + 0,00 | | 98,95%+1,15 | |
| *nursery* | *priority* | 697 | 154 | 837 | 16 | 832 | 21 | 853 | 76 |
| | *spec_priori* | 739 | 70 | 804 | 5 | 792 | 17 | 745 | 0 |
| | *recommend* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | *not_recom* | 864 | 0 | 864 | 0 | 864 | 0 | 863 | 0 |
| | *very_reco* | 39 | 27 | 62 | 4 | 51 | 15 | 53 | 0 |
| | Total | 2339 | 251 | 2567 | 25 | 2539 | 53 | 2513 | 76 |
| | Não | 2 | | 0 | | 0 | | 0 | |
| | Acertos | 90,24% + 0,00 | | 98,53% + 0,00 | | 98,26% + 0,00 | | 97,06%+0,75 | |

A Figura 6.3, sumariza graficamente o desempenho e número de regras geradas de cada sistema. Esta figura está organizada da seguinte forma:

- O eixo *x* à esquerda representa o desempenho global de cada sistema confrontando as regras geradas pelo conjunto de treinamento com o conjunto de teste (as barras do gráfico), os valores deste eixo iniciam em 50% para a escala do gráfico ser mais precisa. O da direita representa o número médio de regras geradas por cada sistema em cada conjunto de dados (as linhas do gráfico); e
- O eixo *y* representa cada conjunto de dados.

Analisando esta figura, observou-se que o DBILP se compara com os sistemas CN2 e C4.5, e é superior ao Progol. Em relação ao número de regras, todos os sistemas foram semelhantes, exceto quando foram classificar o conjunto *nursery*, onde o número de regras geradas por todos foi grande, mas C4.5 e DBILP tiveram piores resultados.

Talvez o desempenho não tão eficiente para o conjunto *nursery* se deve ao fato desse conjunto possuir muitos registros inconsistentes, principalmente aqueles que descrevem a classe *priority*.

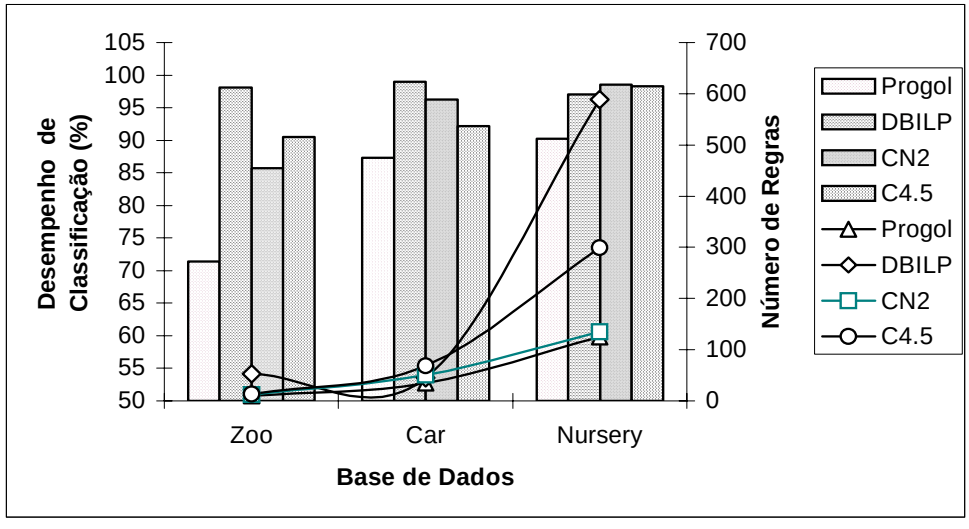



Figura 6.3 – Desempenho de Classificação e Número de Regras Geradas

O trabalho anterior[LOP00] também verificou o desempenho do tempo de processamento para encontrar as regras de cada sistema. Como DBILP não foi executado na mesma plataforma dos sistemas utilizado, este quesito não foi comparado. Contudo observou-se que o DBILP teve um desempenho satisfatório para encontrar as regras. Para descrever o conjunto *car*, o sistema levou menos de cinco minutos (conjunto executado em um Pentium MMX 233, com 94Mb de RAM, sistema operacional Windows NT 4.0, Service Pack 6). O mesmo ambiente demorou, aproximadamente, 20 horas para o conjunto *nursery* e 16 horas para o conjunto *zoo* (apesar do conjunto *zoo* ser o menor de todos o tempo de execução foi superior ao conjunto *car* pelo fato do primeiro possuir 16 candidatos ao corpo, aumentando o número de combinações realizadas).

## 6.4 Considerações Finais

Este capítulo apresentou os resultados de um experimento para verificar o desempenho do DBILP em relação a outros conhecidos sistemas no meio acadêmico, C4.5, CN2 e Progol. Baseado

nesses resultados, foram comparadas diferentes características de desempenho de cada sistema, tais como desempenho de classificação e curva de aprendizado.

Foi observado que DBILP, CN2 e C4.5 tiveram um comportamento bastante similar. Isto foi confirmado quando foi realizado a segunda parte do experimento.

Progol apresentou o pior resultado em relação à curva de aprendizado e ao desempenho de classificação. Isso pode demostrar que Progol é bastante sensível à ordem dos exemplos.

Tanto na curva de aprendizado quanto no desempenho de classificação, DBILP provou ser similar ou até superior aos sistemas comparados. O único item que DBILP não teve um bom desempenho foi na geração do número de regras.

Em sistemas de *data mining* o aumento de alguns pontos percentuais no desempenho de um sistema em relação a outros pode significar uma grande vantagem para as empresas pois pode levar a uma econômia grande de dinheiro, caso a aplicação for em uma bolsa de valores, ou a preservação de vidas, caso a aplicação for em ambientes médicos ou farmacéuticos. Outro ponto importante é, caso um sistema tenha um desempenho de 70% e um segundo chegar a 80% é menos relevante de que um sistema ter um desempenho de 97% e um segundo chegar a 97,5% pois conforme a escala se aproxima de 100% torna-se mais difícil aumentar o grau de acerto. Portanto, mesmo que o DBILP tenha sido um pouco superior aos outros sistemas na maioria do casos, indica que foi um ganho bastante alto pois os sistemas mais eficazes (C4.5 e CN2) estavam, na maioria das vezes, acima de 90%.

A análise e os testes feitos até agora já permitem tirar algumas conclusões, e indicam muitos caminhos para trabalhos futuros, como apresentado no próximo capítulo.

# Capítulo 7

## 7 Conclusões

Neste trabalho foi apresentado DBILP, um sistema para mineração de dados utilizando o poder das técnicas ILP e de SGBD, neste caso, através de comandos SQL. O resultado obtido usando o sistema proposto indica que DBILP é particularmente útil para as atividades de KDD, principalmente porque utiliza-se de um SGBD para o acesso aos exemplos (tuplas), diminuindo a limitação no número de tuplas envolvidas na tarefa de aprendizagem. Assim, DBILP, além de ter um bom comportamento na atividade de aprendizado em relação aos sistemas CN2, C4.5 e Progol, pode ser aplicado em bancos com grandes volumes de dados pois a utiliza os recursos de um SGBD.

Percebeu-se que a linguagem de entrada possui um papel muito importante para o desempenho do sistema DBILP pois um banco de dados pode possuir muitas tabelas e a busca por conceitos sem uma definição de onde buscar e o que induzir se torna bastante difícil. Como o usuário pode informar os relacionamentos entre as tabelas, DBILP torna-se mais flexível pois podem ser usados outros atributos para relacionar as tabelas, que não sejam apenas as chaves estrangeiras.

A linguagem de entrada também permite que regras ingênuas não fossem geradas já que a cabeça e os corpos dos conceitos que podem ser combinados são informados pelo usuário *a priori*, evitando assim busca desnecessárias no banco de dados. Portanto, pode-se concluir que os mecanismos de tendências são bastante importantes no processo de aprendizado já que reduzem o espaço de busca.

Existe bastante estudo para melhorar DBILP. A sua primeira versão tem algumas limitações que podem ser melhoradas com novos estudos. Entre essas limitações podemos apontar:

- O número elevado de regras encontradas quando o conjunto de dados apresentam ruídos, ou seja, linhas inconsistentes que podem dificultar o entendimento do domínio descrito;
- A busca exaustiva das regras. Para viabilizar esta busca, todos os corpos candidatos são utilizados, provocando um aumento exponencial nas tentativas de encontrar as regras. Isto causa uma limitação no número de candidatos, pois, se existirem 16 candidatos, o número de combinações será de 65.535. Este número influi diretamente no desempenho

de processamento do sistema. No caso da base *zoo* que possui apenas 80 linhas porém 16 candidatos ao corpo da regra, o sistema demorou quase 16h para descobrir todas as regras (conjunto executado em um Pentium MMX 233, com 94Mb de RAM, sistema operacional Windows NT 4.0, Service Pack 6). Porém no conjunto *car* que possui 1.382 linhas e 6 candidatos ao corpo da regra, o tempo de execução foi inferior à 5 minutos.

Embora os experimentos realizados neste trabalho tenham usado bases de dados pequenas, que cabem na memória principal do computador, o uso de SQL permite que o sistema trabalhe diretamente com grandes volumes de dados, que não cabem na memória principal. Isso deverá ser avaliado em pesquisas futuras.

De qualquer modo cabe ressaltar que o uso de SQL para acessar os dados minerados, que são mantidos dentro de um SGBD durante toda a execução do algoritmo de *data mining*, também oferece outras vantagens, tais como:

- re-uso de dados e minimização de redundância dos mesmos;
- melhor segurança e controle de privacidade dos dados;
- potencial para re-uso de *software* (consultas SQL semelhantes podem ser utilizadas por algoritmos diferentes); e
- melhor capacidade de expansão (*scalability*) para grande conjunto de dados.

Assim, DBILP mostrou-se bastante eficaz na atividade de geração de regras de primeira ordem para classificação de dados em banco de dados relacionais. As limitações e deficiências apresentadas, além de melhorias, podem ser implementadas com alguns estudos mais profundos. Estes ajustes são citados na próxima seção.

## 7.1 Trabalhos Futuros

Baseado no que foi descrito na seção anterior, têm-se as seguintes perspectivas de continuação desse trabalho:

- Utilizar heurística para escolher os candidatos ao corpo da regra candidata;
- Utilizar heurística também para limitar o número de linhas das tabelas envolvidas na mineração para a descoberta das regras candidatas;
- Colocar um parâmetro de fim de pesquisa quando uma determinada classe gerar muitas regras candidatas;

- Definir parâmetros de confiança e suporte para otimizar a busca das regras;
- Permitir recursividade na definição das regras;
- Permitir que a cabeça da regra tenha uma aridade diferente de binária;
- Definir na linguagem regras que podem ser descartadas como candidatas antes de serem testadas, e.g., *avo (A,X):-avo(A,X)*. Atualmente é permito apenas a declaração dos corpos e da cabeça, sem a inclusão de um conhecimento prévio do domínio para algumas combinações;
- Permitir a definição de mais de uma cabeça candidata à regra;
- Permitir utilizar um corpo negado como parte de uma regra;
- Aceitar alguns exemplos positivos e/ou negativos informados na linguagem de definição, não estando presentes, necessariamente, no banco de dados; e
- Realizar experimentos com volumes de dados bem maiores,que não podem ser mantidos na memória principal.

Portanto existe bastante trabalho para melhorar a primeira versão do sistema DBILP. Mesmo assim, DBILP tem duas grandes qualidades: usa o poder de expressão de ILP para induzir conceitos, e explora o poder e a segurança dos SGBD para minerar dados.

## Bibliografia


[ABI95] S. Abiteboul, R. Hull and V. Vianu. *Foundations of Databases*. Addison-Wesley Publishing Company, 1995.

[ADR97] P. Adriaans and D. Zantinge. *Data Mining*. Addison Wesley Longman, 1st. Edition, 1997.

[AGR97] R. Agrawal, A. Gupta and S. Sarawagi. *Modeling Multidimensional Databases. Research Report*. IBM Research Division. ICDE 1997: 232-243, 1997

[ARU96] M. Arunasalam. *Eletronic Book: Advanced DataBase Systems - Chapter 2 Data Mining*. http://www.rpi.edu/~arunmk/dm1.html. 1996.

[BAR94] R. Barker. *Case*Method*sm *– Entity Relationship Modelling*. Addilson Wesley. Wokinghgam, England. 1994.

[BER96] F. Bergano and Daniele Gunetti. *Inductive Logic Programming: From Machine Learning to Software Engineering*. The Mit Press, 1996.

[BLA98] C. L. Blake and C.J. Merz. *UCI Repository of Machine Learning Data Bases*. Disponível em <http://www.ics.uci.edu/~mlearn/MLRepository.html> Irvine, CA: University of California, Departament of Information and Computer Science, 1998.

[BLO96] H. Blockeel and L. De Raedt. *Relational Knowledge Discovery in Database*. 6th International Workshop on Inductive Logic Programming – ILP96.

[BOS98] H. Boström, *Predicate Invention and Learning from Positive Exanples Only*. Proc. of the Tenth European Conference on Machine Learning, Springer Verlag (1998) 226-237

[BRA98] I. Bratko, M. Kubat and R. Michalski. *Machine Learning and Data Mining*. John Wiley & Sons Ltd., 1st Edition, 1998.

[BRO96] P. Brockhausen and K. Morik. Direct *Access of an ILP Algorithm to a Database Management System*. Data Mining with Inductive Logic Programming (ILP for KDD), Pfaninger, Bernhand and Fürnkranz, Joahannes (ed), Mlnet Sponsored Familiarization Workshop, 1996.

[CHA96] S. Chaudhuri and U. Dayal. *An Overview of Data Warehousing and OLAP Technology*. Sigmod Record , March 1997 (with Umesh Dayal). Tutorials Presented at 1996 VLDB, 1997 SIGMOD, 1998 EDBT and 1998 IEEE ICDE Conferences

[CLA91] P. Clark and R. Boswell. *Rule Induction with CN2: Some Recent Improvements.* Proceedings of the Fifth European Working Session on Learning, Berlin(1991), 151-163.

[DAT89] C. J. Date. *Introdução a Sistemas de Banco de Dados*. Editora Campus, 7ª. Reimpressão, 1989. Rio de Janeiro.

[DAV95] S. Davidson, P. Buneman and A. Kosky. *Semantics of Database Transformations.* Technial Report MS-CIS-95-25, University of Pennsylvania, 1995.

[DEV97] B. Devlin. *Data Warehouse: From Architecture to Implementation*. Addison Wesley Longman, 1st Edition, 1997.

[DZE96] S. Dzeroski. *Inductive Logic Programming and Knowledge Discovery in Database. Advances Discovery and Data Mining*. Menlo Park. CA:AAAI/MIT, pp 118-151, 1996.

[DZE93] S. Dzeroski. *Handling Imperfect Data in Inductive Logic Programming*. In Proc. Fourth Scandinavian Conference on Artificial Intelligence, pages 111--125, IOS Press, Amsterdam, 1993.

[FAY98] A. Faye, A. Giacometti, D. Laurent and N. Spyratos. *Mining Significant Rules from Databases*. Networking and Information Systems Journal, Vol. 1, No. 1, pp. 653-682.

[GRA96] W. K. Grassmann and Jean-Paul Tremblay. *Logic and Discrete Mathematics – A Computer Science Perspective*. Prentice Hall, New Jersey, 1996.

[HAM97] J. Hammer, H. Garcia-Molina, S. Nestorov, R. Yerneni and M. Breuning e V. Vassalos. *Template-Based Wrappers in the TSIMMIS System*. In Proceedings of the Twenty-Sixth SIGMOD International Conference on Management of Data, Tucson, Arizona, May 12-15, 1997, 1997.

[INM97] W. H. Inmon e R. D. Hackathorn. *Como Usar o Data Warehouse*. Livraria e Editora Infobook, 1ª Edição, 1997.

[KIE92] Jörg-Uwe Kietz and Stefan Wrobel. *Controlling the Complexity of Learning in Logic Through Syntatic and Task-Oriented Models*. In Stephen Muggleton, editor, Inductive Logic Programming, Chapter 16, pages 335-360. Academic Press, London, 1992.

[LAV94] N. Lavrac and Saso Dzeroski. *Inductive Logic Programming – Techniques and Applications*. Ellis Horwood, New York, 1994.

[LOP00] F. Lopes, A. Pozo, S. Vergílio e D. Duarte. *Sistemas Baseados em Indução: uma comparação empírica*. Memorias del X CLAIO (Congreso Latino-Iberoamericano de Investigación de Operaciones y Sistemas), Mexico, Mexico. Septiembre del 2000.

[MAN97] H. Mannila. *Methods and Problems in Data Mining*. Proceedings of International Conference on Database Theory (ICDT'97), Delphi, Greece, January 1997, F. Afrati and P. Kolaitis (ed.), p. 41-55.

[MIC83] R. Michalski. *A Theory and Methodology of Inductive Logic Programming.* In Michalski, R.S., Carbonell, J.G. e Mitchell, T.M, editors, *Machine Learning: Na Artificial Intelligence Approach*, Volume I, pages 83-134. Tioga, Palo Alto, CA.

[MOR93] K. Morik et al, *Knowledge Acquisition and Machine Learning: Theory, Methods and Applications*. London: Academic Press, 1993.

[MOR97] K. Morik. *Knowledge Discovery in Database – An Inductive Logic Programming Approach*. Foundations of Computer Science – Theory, Cognition, Applications. Fuska, Jantzer and Valk (ed.). Lecture Notes in Computer Science, 1997.

[MOT97] R. Motwani. *Position for UW*. Microsoft Summer Research Institute on Data Mining, 1997.

[MUG90] S.Muggleton and W.Buntime. *Efficient Induction of Logic Programs*. In Proc. of the First Conf. on Algorithmic Learning Theory, Tokyo, 1990.

[MUG94] S. Muggleton and L. De Raedt. *Inductive Logic Programming: Theory and Methods*. Journal of Logic Progamming. pp. 19/20:629-679, 1994.

[MUG95] S. Muggleton. *Inverse Entailment and Progol*. New Generation Computing, 13:245-286, 1995.

[MUG97] S. Muggleton. *Declarative Knowledge Discovery in Industrial Databases*. In H.F. Arner, editor, Proceedings of the First International Conference and Exhibition on The Practical Application of Knowledge Discovery and Data Mining (PADD-97), pages 9-24. Practical Application Company Ltd., 1997

[MUG00] S. Muggleton and J. Firth. *CProgol4.4: Theory and Use*.

[MUS97] N. H. Mustafa. *Inductive Logic Programming : A brief introduction to Theory, and ILP Systems*.

[OLI98] A. G. de Oliveira. *Data Warehouse - Conceitos e Soluções*. Advanced Editora, 1ª Edição, 1998.

[QUA96] D. Quass and J. Widom. *On-Line View Maintenance for Batch Updates*. SIGMOD '97, 1996.

[QUI86] J. Quinlan. *Inductive of Decision Trees*. Machine Learning, 1(1):81-106.

[QUI90] J.R. Quinlan. *Learning Logical Definitions From Relations*. Machine Learning, 5:239-266, 1990.

[QUI93] J. Quinlan. *C4.5: Programs for Machine Learning*, Morgan Kaufmann, 1993.

[RAE92] L. De Raedt, *Interactive Theory Revision: An Inductive Logic Programming Approach*. London: Academic Press. 1992

[RAE93] L. De Raedt and M. Bruynooghe. *A Theory of Clausal Discovery*. In Stephen Muggleton, editor, Procs. of the 3rd. International Workshop on Inductive Logic Programming, number IJS-DP-6707 in J. Stefan Institute Technical Reports, pages 25-40,1993.

[RAE95] L. De Raedt and L. Dehaspe. *Clausal Discovery*. Forthcoming, 1995.

[ROB97] S. Roberts. *An Introduction to Progol*. January 21, 1997.

[RUS95] S. Russel and P. Norvig. *Artificial Intellligence : A Modern Approach*. Prentice-Hall, Inc., 1st Edition, 1995.

[SHA83] E. Y. Shapiro. *Algorithmic Program Debugging*. MIT Press, 1983.

[SHI97] K. Shimazu and K. Furukawa. *Knowledge Discovery in Database by PROGOL – Design, Implementation and its Application to Expert System Building*. Proceedings

of The First International Conference on The Practical Application of Knowledge Discovery and Data Mining (PADD97). April 1997

[TWC99] Two Crows Corporation.. *Introduction to Data Mining and Knowledge Discovery*. Third Edition (October 8, 1999). Two Crows Corporation.

[UBE00] Queen's University of Belfast - Parallel Computer Centre. *Data Mining: An Introduction - Student Notes*. http://www.pcc.qub.ac.uk/tec/courses/datamining/stu_notes/dm_book_1.html em 12/10/2000 às 14:30h

[ULM97] J. Ulmann. *Position for UW*. Microsoft Summer Research Institute on Data Mining, 1997.

[WEI91] S. M. Weiss and C. A. Kulikowski. *Computer Systems That Learn*. Morgan Kaufman, 1991.

[WEI98] S. M. Weiss and N. Indurkhya. *Predictive Data Mining: A Pratical Guide*. Morgan Kaufmann Publishers, Inc., 1st Edition, 1998.

[WID95] J. Widom. *Research Problems in Data Warehousing*. Proc. Of 4Th Int'l Conference on Information and Knowledge Management (CIKM), Nov. 1995.

[WID97] J. D. Ulmman and J. Widom. A First Course In Database Systems. Prantice Hall, 1997.

[ZHU95] Y. Zhuge, H. Garcia-Molina, J. Hammer and J. Widom. *View Maintence in a Warehousing Environment*. Proceedings of the ACM SIGMOD International Conference on Management of Data, San Jose, California, June 1995, 1995.

# Apêndices

## Apêndice A - Algoritmos

### A.1 Algoritmo Encontra Hipóteses Candidatas

```
Algorithm HypothesisFinder
   Input: View
   Input: BodiesCombinantion
   Input: BodiesCandidates
   Input: HypothesisHead
   Output: SetHypothesisCandidates
   i,j ←0
   Vector [i,j]← ClassesCountFromView
   While Vector[i,0] != null Do
      Row ← NextRow(Vector[i,0])
      While Exists(Row) Do
         ActCombination ← NextCombination(BodiesCombinantion)
         While Exists (ActCombination)Do
              Where ← CreateWhereClause(ActCombination, Row)
              QtyPositive ← CountPositiveExamples(Where, Vector[i,0])
              QtyNegative ← CountNegativeExamples(Where, Vetor)
              If QtyNegative == 0
                 If QtyPositive == Vector[i,1]
                    SetHypothesisCandidates [i]←null
                    SetHypothesisCandidates [i,0] ← GetHypothesis(Where)
                    ActCombinationcombina ← null
                    Row ← null
                    Break to Third While
                 End If
                 SetHypothesisCandidates [i,m] ← GetHypothesis(Where)
                 m++
              End If
              ActCombination ← NextCombination(BodiesCombinantion)
         End While
         Row ← NextRow(Vector[i,0])
      End While
      i++
   End While
End HypothesisFinder
```

## A.2 Algoritmo Retira Hipóteses subjugadas

```
Algorithm DropHypothesis
   Input: SetHypothesisCandidates
   Output: SetHypothesis
   OrderSet(SetHypothesisCandidates,SetHypothesis,byLength)
   int i←0
   String Candidate← SetHypothesis[i]
   While SetHypothesis[i]!=null Do
       For j=i+1 To SetHypothesis.length Do
          If Candidate Subssumes SetHypothesis[j]
             DropElement(SetHypothesis, SetHypothesis[j])
          End If
       End For
       i++
       Candidate← SetHypothesis[i]
   End While
End Hypothesis
```